COURS et TP DE LANGAGE C++

Chapitre 13

Surcharge des opérateurs

Joëlle MAILLEFERT

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

IUT de CACHAN

Département GEII 2

CHAPITRE 13

SURCHARGE DES OPERATEURS

I- INTRODUCTION

Le langage C++ autorise le programmeur à étendre la signification d'opérateurs tels que l'addition (+), la soustraction (-), la multiplication (*), la division (/), le ET logique (&) etc...

Exemple:

On reprend la classe vecteur déjà étudiée et on surdéfinit l'opérateur somme (+) qui permettra d'écrire dans un programme:

vecteur v1, v2, v3;
v3 = v2 + v1;

Exercice XIII-1:

Etudier et tester le programme suivant:

```
#include <iostream.h>
#include <conio.h>
// Classe vecteur : surcharge de l'opérateur +

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float abs,float ord);
      void affiche();
    // surcharge de l'opérateur somme, on passe un paramètre vecteur
    // la fonction retourne un vecteur
      vecteur operator + (vecteur v);
   };

    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;
}

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

Exercice XIII-2:

Ajouter une fonction membre de prototype **float operator*(vecteur v)** permettant de créer l'opérateur « produit scalaire », c'est à dire de donner une signification à l'opération suivante:

vecteur v1, v2;
float prod_scal;
prod_scal = v1 * v2;

Exercice XIII-3:

Ajouter une fonction membre de prototype **vecteur operator*(float)** permettant de donner une signification au produit d'un réel et d'un vecteur selon le modèle suivant :

vecteur v1,v2;
float h;
v2 = v1 * h ; // homothétie

Les arguments étant de type différent, cette fonction peut cohabiter avec la précédente.

Exercice XIII-4:

Sans modifier la fonction précédente, essayer l'opération suivante et conclure.

vecteur v1,v2;
float h;
v2 = h * v1; // homothétie

Cette appel conduit à une erreur de compilation. L'opérateur ainsi créé, n'est donc pas symétrique. Il faudrait disposer de la notion de « fonction amie » pour le rendre symétrique.

II- APPLICATION: UTILISATION D'UNE BIBLIOTHEQUE

BORLAND C++ possède une classe « complex », dont le prototype est déclaré dans le fichier **complex.h**.

Voici une partie de ce prototype:

```
 class complex
{
private:
   double re,im; // partie réelle et imaginaire d'un nombre complexe
public:
   complex(double reel, double imaginaire = 0); // constructeur
 // complex manipulations
   double real(complex);   // retourne la partie réelle
   double imag(complex);   // retourne la partie imaginaire
   complex conj(complex);  // the complex conjugate
   double norm(complex);   // the square of the magnitude
   double arg(complex);    // the angle in radians
 // Create a complex object given polar coordinates
   complex polar(double mag, double angle=0);
 // Binary Operator Functions
   complex  operator+(complex);
 // donnent un sens à : « complex + double » et « double + complex »
   friend complex  operator+(double, complex);
   friend complex  operator+(complex, double);
 // la notion de « fonction amie » sera étudiée au prochain chapitre
   complex  operator-(complex);
   friend complex  operator-(double, complex);
 // idem avec la soustraction
   friend complex  operator-(complex, double);
   complex  operator*(complex);
   friend complex  operator*(complex, double);
 // idem avec la multiplication
   friend complex  operator*(double, complex);
   complex  operator/(complex);
   friend complex  operator/(complex, double);
 // idem avec la division
   friend complex  operator/(double, complex);
   int  operator==(complex);          // retourne 1 si égalité
   int  operator!=(complex, complex); // retourne 1 si non égalité
   complex  operator-();              // opposé du vecteur
};
```

```
// Complex stream I/O
// permet d'utiliser cout avec un complexe
   ostream    operator<<(ostream  , complex);
// permet d'utiliser cin avec un complexe
   istream    operator>>(istream  , complex);
```

Exercice XIII-5:
Analyser le fichier **complex.h** pour identifier toutes les manipulations possibles avec les nombres complexes en BORLAND C++.
Ecrire une application.

III- REMARQUES GENERALES (***)

- Pratiquement tous les opérateurs peuvent être surdéfinis:
+ - * / = ++ -- new delete [] -> & | ^ && || % << >> etc ...
avec parfois des règles particulières non étudiées ici.
- Il faut se limiter aux opérateurs existants.
- Les règles d'associativité et de priorité sont maintenues.
- Il n'en est pas de même pour la commutativité (cf exercice XIII-3 et XIII-4).
- L'opérateur = peut-être redéfini. S'il ne l'est pas, une recopie est exécutée comme on l'a vu dans le chapitre II (cf exercice II-1, à re-tester).
Si la classe contient des données dynamiques, il faut impérativement surcharger l'opérateur =.

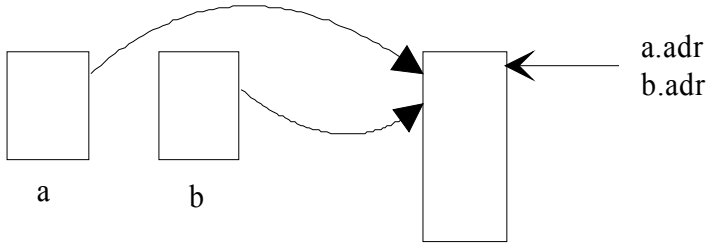
Exercice XIII-6:

Dans le programme ci-dessous, on surdéfinit l'opérateur =.
En étudier soigneusement la syntaxe, tester avec et sans la surcharge de l'opérateur =.
Conclure.

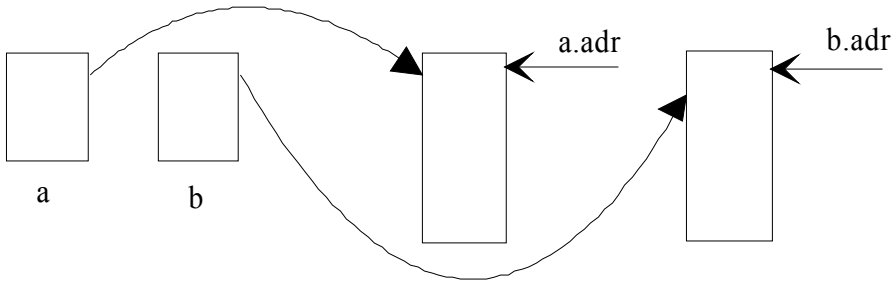
```
#include <iostream.h>
#include <conio.h>

    class liste
   {
   private:
      int taille;
      float *adr;
   public:
      liste(int t);            // constructeur
      liste::liste(liste &v);  // constructeur par recopie
      void saisie(); void affiche();
      void operator=(liste &); // surcharge de l'opérateur =
      ~liste();
   };
```

```
 liste::liste(int t)
{
   taille = t;adr = new float[taille];cout<<"Construction";
   cout<<" Adresse de l'objet:"<< this;
   cout<<" Adresse de liste:"<< adr <<"\n";
}

 liste::~liste()
{
   cout<<"Destruction Adresse de l'objet:"<< this;
   cout<<" Adresse de liste:"<< adr <<"\n"; delete adr;
}

 liste::liste(liste &v)
{
   taille = v.taille;adr = new float[taille];
   for(int i=0;i<taille;i++)adr[i] = v.adr[i];
   cout<<"\nConstructeur par recopie";
   cout<<"  Adresse de l'objet:"<< this;
   cout<<"  Adresse de liste:"<< adr <<"\n";}

 void liste::saisie()
{
   for(int i=0;i<taille;i++)
   {cout<<"Entrer un nombre:";cin>>*(adr+i);}
}

 void liste::affiche()
{
   cout<<"Adresse:"<<this<<" ";
   for(int i=0;i<taille;i++)cout<<*(adr+i)<<" ";
   cout<<"\n\n";
}

 void liste::operator=(liste &lis)
{/* passage par référence pour éviter l'appel au constructeur par
 * recopie et la double libération d'un même emplacement mémoire */
   taille=lis.taille; delete adr; adr=new float[taille];
   for(int i=0;i<taille;i++)adr[i] = lis.adr[i];
}

 void main()
{
   cout<<"Debut de main()\n";
   liste a(5); liste b(2);
   a.saisie();  a.affiche();
   b.saisie();  b.affiche();
   b = a;
   b.affiche(); a.affiche();
   cout<<"Fin de main()\n";
}
```

On constate donc que la surcharge de l'opérateur = permet d'éviter la situation suivante:



et conduit à:



Conclusion:

Une classe qui présente une donnée membre allouée dynamiquement doit toujours posséder au minimum, un constructeur, un constructeur par recopie, un destructeur, la surcharge de l'opérateur =. Une classe qui possède ces propriétés est appelée « classe canonique ».

IV- EXERCICES RECAPITULATIFS (***)

Exercice XIII-7:

Reprendre la classe pile_entier de l'exercice IV-13 et remplacer la fonction membre « empile » par l'opérateur < et la fonction membre « depile » par l'opérateur >.
p < n ajoute la valeur n sur la pile p
p> n supprime la valeur du haut de la pile p et la place dans n.

Exercice XIII-8:

Ajouter à cette classe un constructeur par recopie et la surdéfinition de l'operateur =

Exercice XIII-9:

Ajouter à la classe **liste** la surdéfinition de l'opérateur [], de sorte que la notation a[i] ait un sens et retourne l'élément d'emplacement i de la liste a.
Utiliser ce nouvel opérateur dans les fonctions **affiche** et **saisie**
On créera donc une fonction membre de prototype ***float &liste::operator[](int i);***

Exercice XIII-10:

Définir une classe **chaine** permettant de créer et de manipuler une chaîne de caractères:
données:

- longueur de la chaine (entier)
- adresse d'une zône allouée dynamiquement (inutile d'y ranger la constante \0)

méthodes:

- constructeur **chaine()** initialise une chaîne vide
- constructeur **chaine(char *texte)** initialise avec la chaîne passée en argument
- constructeur par recopie **chaine(chaine &ch)**
- opérateurs affectation (=), comparaison (==), concaténation (+), accès à un caractère de rang donné ([])

V- CORRIGE DES EXERCICES

Exercice XIII-2:

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

// Classe vecteur, surcharge de l'opérateur produit scalaire

    class vecteur
   {
   private:
      float x,y;
   public: vecteur(float abs,float ord);
      void affiche();
      vecteur operator+(vecteur v); // surcharge de l'operateur +
    // surcharge de l'opérateur * en produit scalaire
      float operator*(vecteur v);
   };

    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;
   }

    void main()
   {
      vecteur a(2,6),b(4,8),c;
      float prdscl1,prdscl2,prdscl3;
      c = a + b; c.affiche();
      prdscl1 = a * b;
      prdscl2 = a.operator*(b);
      prdscl3 = b.operator*(a);
      cout<<prdscl1<<" "<<prdscl2<<" "<<prdscl3<<"\n"; getch();
   }
```

Exercice XIII-3:

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

    class vecteur
   {
   private:
      float x,y;
   public: vecteur(float abs,float ord);
      void affiche();
      vecteur operator+(vecteur v); // surcharge de l'opérateur +
      float operator*(vecteur v);   // surcharge de l'opérateur *
                                    // produit scalaire
      vecteur operator*(float f);   // surcharge de l'opérateur *
                                    // homothétie
   };

    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)
   {
      return v.x * x + v.y * y;
   }

    vecteur vecteur::operator*(float f)
   {
      vecteur res; res.x = f*x;   res.y = f*y;
      return res;
   }

    void main()
   {
      vecteur a(2,6),b(4,8),c,d;
      float prdscl1,h=2.0;
      c = a + b; c.affiche();
      prdscl1 = a * b; cout<<prdscl1<<"\n";
      d =  a * h; d.affiche();getch() ;
   }
```

Exercice XIII-5:

```
#include <iostream.h>  // Utilisation de la bibliothèque
#include <conio.h>     // de manipulation des nombres complexes
#include <complex.h>

#define PI 3.14159

    void main()
   {
      complex a(6,6),b(4,8),c(5);
      float n =2.0,x,y;
      cout<<"a = "<< a <<" b= "<< b <<" c= "<< c <<"\n" ;
      c = a + b;              cout<<"c= "<< c <<"\n" ;
      c = a * b;              cout<<"c= "<< c <<"\n" ;
      c = n * a;              cout<<"c= "<< c <<"\n" ;
      c = a * n;              cout<<"c= "<< c <<"\n" ;
      c = a/b;                cout<<"c= "<< c <<"\n" ;
      c = a/n;                cout<<"c= "<< c <<"\n" ;
      x = norm(a); cout<<"x= "<< x <<"\n" ;
      y = arg(a)*180/PI;  // Pour l'avoir en degrés
      cout<<"y= "<< y <<"\n" ;
      c = polar(20,PI/6); // module = 20  angle = 30°
      cout<<"c= "<< c <<"\n" ;
      c = -a;   cout<<"c= "<< c <<"\n" ;
      c = a+n;  cout<<"c= "<< c <<"\n" ;
      cout<< (c==a) <<"\n" ;
      cout<<"Saisir c sous la forme (re,im): ";
      cin >> c;
      cout<<"c= "<< c <<"\n" ;
      getch();
   }
```

Exercice XIII-7:

```
    class  pile_entier
   {
   private :
      int *pile,taille,hauteur;
   public:
      pile_entier(int n); // constructeur, taille de la pile
      ~pile_entier();     // destructeur
      void operator < (int x);  // ajoute un élément
      void operator >(int &x);  // dépile un élément
      int pleine();             // 1 si vrai 0 sinon
      int vide();               // 1 si vrai 0 sinon
   };
```

```
 pile_entier::pile_entier(int n=20) // taille par défaut: 20
{
   taille = n;
   pile = new int[taille];  // taille de la pile
   hauteur = 0;
   cout<<"On a fabriqué une pile de "<< taille <<" éléments\n";
}

 pile_entier::~pile_entier()
{
   delete pile;// libère la mémoire
}

 void pile_entier::operator<(int x)
{
   *(pile+hauteur) = x; hauteur++;
}

 void pile_entier::operator >(int &x)
{ // passage par référence obligatoire (modification de l'argument)
   hauteur--; x = *(pile+hauteur);
}

 int pile_entier::pleine()
{
   if(hauteur==taille)
      return 1;
   return 0;
}

 int pile_entier::vide()
{
   if(hauteur==0)
      return 1;
   return 0;
}


 void main()
{
   pile_entier a ;
   int n = 8,m;
   a < n;
   if (a.vide()) cout<<"La pile est vide\n";
   else cout<<"La pile n'est pas vide\n";
   a > m;
   cout<<"m="<< m <<"\n";
   if (a.vide()) cout<<"La pile est vide\n";
   else cout<<"La pile n'est pas vide\n";
   getch();
}
```

Exercice XIII-8: On ajoute les éléments suivants:

```
#include <iostream.h> // Gestion d'une pile d'entiers
#include <conio.h>

    class  pile_entier
   {
   private :
      int *pile,taille,hauteur;
   public:
      pile_entier(int n); // constructeur, taille de la pile
      pile_entier(pile_entier &p); // constructeur par recopie
      ~pile_entier(); // destructeur
      void operator < (int x);  // ajoute un élément
      void operator >(int &x);  // dépile un élément
      void operator = (pile_entier &p); // surcharge de l'operateur =
      int pleine();             // 1 si vrai 0 sinon
      int vide();               // 1 si vrai 0 sinon
   };

    pile_entier::pile_entier(int n=20)  // taille par défaut: 20
   {
      taille = n; // taille de la pile
      pile = new int[taille]; hauteur = 0;
      cout<<"On a fabriqué une pile de "<<taille<<" éléments\n";
   }

    pile_entier::pile_entier(pile_entier &p) // constructeur par recopie
   {
      taille = p.taille;
      pile =  new int [taille] ; hauteur =  p.hauteur;
      for(int i=0;i<hauteur;i++)*(pile+i) = p.pile[i];
      cout<<"On a bien recopié la pile\n";
   }

    pile_entier::~pile_entier()
   {
      delete pile; // libère la mémoire
   }

    void pile_entier::operator<(int x)
   {
      *(pile+hauteur) = x; hauteur++;
   }

    void pile_entier::operator >(int &x)
   {// passage par référence obligatoire (modifier valeur de l'argument)
      hauteur--; x = *(pile+hauteur);
   }
```

```
 int pile_entier::pleine()
{
   if(hauteur==taille)
      return 1;
   return 0;
}

 int pile_entier::vide()
{
   if(hauteur==0)
      return 1;
   return 0;
}

 void pile_entier::operator = (pile_entier &p)
{
   taille = p.taille;
   pile = new int [taille];
   hauteur = p.hauteur;
   for(int i=0;i<hauteur;i++)
      *(pile+i)=p.pile[i];
   cout<<"l'égalité, ça marche !\n";
}

 void main()
{
   pile_entier a,c(10);
   int m,r,s ;
   for(int n=5;n<22;n++)
      a < n;            // empile 18 valeurs
   pile_entier b = a;
   for(int i=0; i<3;i++)
   {
      b>m;cout<<m<<" "; // dépile  3 valeurs
   }
   cout<<"\n";c = a;
   for(int i=0;i<13;i++)
   {
      c>r;cout<<r<<" "; // dépile 13 valeurs
   }
   cout<<"\n";
   for(int i=0; i<4;i++)
   {
      a>s;cout<<s<<" "; // dépile  4 valeurs
   }
   getch();
}
```

Exercice XIII-9:

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

    class liste   // NOTER LA MODIFICATION DES FONCTIONS
    // Fonctions saisie ET affiche QUI UTILISENT L'OPERATEUR []
   {
   private:
      int taille;
      float *adr;
   public:
      liste(int t);
      liste(liste &v);
      void operator=(liste &lis); // surcharge de l'opérateur =
      float &operator[](int i); // surcharge de l'opérateur []
      void saisie();
      void affiche();
      ~liste();
   };

    liste::liste(int t)
   {
      taille = t;adr = new float[taille];cout<<"Construction";
      cout<<" Adresse de l'objet:"<< this;
      cout<<" Adresse de liste:"<< adr <<"\n";
   }

    liste::~liste()
   {
      cout<<"Destruction Adresse de l'objet : "<< this;
      cout<<" Adresse de liste : "<< adr <<"\n";
      delete adr;
   }

    liste::liste(liste &v)
   {
      taille = v.taille; adr = new float[taille];
      for(int i=0;i<taille;i++)adr[i] = v.adr[i];
      cout<<"\nConstructeur par recopie";
      cout<<"  Adresse de l'objet : "<< this;
      cout<<"  Adresse de liste : "<< adr <<"\n";
   }

    void liste::operator=(liste &lis)
   { /* passage par référence pour éviter l'appel au constructeur par
     * recopie et la double libération d'un même emplacement mémoire */
      taille=lis.taille;
      delete adr; adr=new float[taille];
      for(int i=0;i<taille;i++) adr[i] = lis.adr[i];
   }
```

```
 float &liste::operator[](int i) // surcharge de []
{
   return adr[i];
}

 void liste::saisie()  // UTILISATION DE []
{
   for(int i=0;i<taille;i++)
   {cout<<"Entrer un nombre:";cin>>adr[i];}
}

 void liste::affiche() // UTILISATION DE []
{
   cout<<"Adresse:"<< this <<" ";
   for(int i=0;i<taille;i++) cout<<adr[i]<<" ";
   cout<<"\n\n";
}

 void main()
{
   cout<<"Début de main()\n";
   liste a(3);
   a[0]=25; a[1]=233;
   cout<<"Saisir un nombre:";
   cin>>a[2];  a.affiche();
   a.saisie(); a.affiche();
   cout<<"Fin de main()\n";
   getch();
}
```

Exercice XIII-10:

```
#include <iostream.h> // classe chaine
#include <conio.h>

 class chaine
{
private :
   int longueur; char *adr;
public:
   chaine();chaine(char *texte);chaine(chaine &ch);  //constructeurs
   ~chaine();
   void operator=(chaine &ch);
   int operator==(chaine ch);
   chaine &operator+(chaine ch);
   char &operator[](int i);
   void affiche();
};
```

```
 chaine::chaine(){longueur = 0;adr = new char[1];} //constructeur1

 chaine::chaine(char *texte) // constructeur2
{
   for(int i=0;texte[i]!='\0';i++);
   longueur = i;
   adr = new char[longueur+1];
   for(int i=0;i!=(longueur+1);i++) adr[i] = texte[i];
}

 chaine::chaine(chaine &ch) //constructeur par recopie
{
   longueur = ch.longueur;
   adr = new char[longueur];
   for(int i=0;i!=(longueur+1);i++)adr[i] = ch.adr[i];
}

 void chaine::operator=(chaine &ch)
{
   delete adr;
   longueur = ch.longueur;
   adr = new char[ch.longueur+1];
   for(int i=0;i!=(longueur+1);i++) adr[i] = ch.adr[i];
}

 int chaine::operator==(chaine ch)
{
   res=1;
   for(int i=0;(i!=(longueur+1))&&(res!=0);i++)
      if(adr[i]!=ch.adr[i])res=0;
   return res;
}

 chaine &chaine::operator+(chaine ch)
{
   static chaine res;
   res.longueur = longueur + ch.longueur;
   res.adr = new char[res.longueur+1];
   for(int i=0;i!=longueur;i++) res.adr[i] = adr[i];
   for(int i=0;i!=ch.longueur;i++)res.adr[i+longueur] = ch.adr[i];
   res.adr[res.longueur]='\0';
   return(res);
}

 char &chaine::operator[](int i)
{
   static char res='\0';
   if(longueur!=0) res =  *(adr+i);
   return res;
}
```

```
 chaine::~chaine()
{
   delete adr;
}

 void chaine::affiche()
{
   for(int i=0;i!=longueur;i++)
   {
      cout<<adr[i];
   }
   cout<<"\n";
}

 void main()
{
   chaine a("Bonjour "),b("Maria"),c,d("Bonjour "),e;
   if(a==b)cout<<"Gagné !\n";
   else    cout<<"Perdu !\n";
   if(a==d)cout<<"Gagné !\n";
   else    cout<<"Perdu !\n";
   cout<<"a: ";  a.affiche();
   cout<<"b: ";  b.affiche();
   cout<<"d: ";  d.affiche();

   c = a+b;
   cout<<"c: ";  c.affiche();

   for(int i=0;c[i]!='\0';i++) cout<<c[i];
   getch();
}
```