Chapitre 5 : Surdéfinition des opérateurs

C++ autorise la surdéfinition des fonctions membres ou indépendantes en fonction du nombre et du type d'arguments.
C++ autorise également la surdéfinition des opérateurs portant au moins sur un objet, tel que l'addition (+), la soustraction (-) ou l'affectation (=) entre objets.

1. Le mécanisme de la surdéfinition des opérateurs

Pour surdéfinir un opérateur 'op', il faut définir une fonction de nom : 'operator op'.

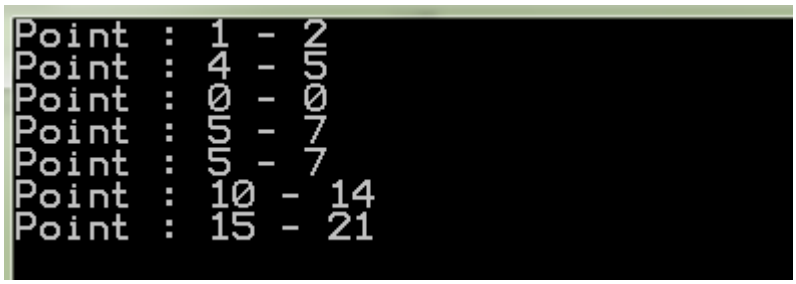
Exemple

```
Point operator + (point,point);
```

1.1. Surdéfinition d'opérateur avec une fonction amie

```
class point
{
       int x,y;
     public:
       point(int a=0,int b=0) {x=a; y=b;}
       void affiche()
       {cout<<"Point : "<<x<<" - " <<y<<endl;}
       friend  point operator + (point,point);
};
//----------------------------------------

point operator + (point p1,point p2)
{
     point p;
     p.x=p1.x+p2.x;
     p.y=p1.y+p2.y;
     return p;
}
//-------------------------

main()
{
     point o1(10,20);          o1.affiche();
     point o2(45,50);          o2.affiche();
     point o3;                 o3.affiche();
     o3=o1+o2;                 o3.affiche();
     o3=operator+(o1,o2);      o3.affiche();
     o3=o1+o2+o3;              o3.affiche();
     o3=operator+(operator+(o1,o2),o3); o3.affiche();
}
```

Exécution



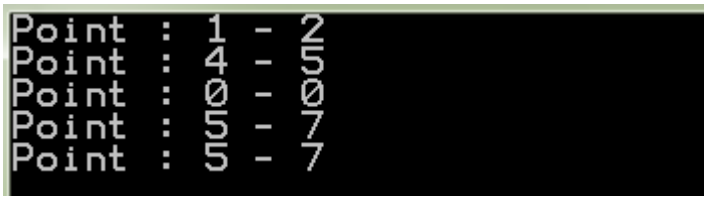
1.2. Surdéfinition d'opérateur avec une fonction membre

L'expression '`o1+o2`' sera interprétée par le compilateur comme l'expression '`o1.operator+(o2);`'.
Le prototype de la fonction membre '`operator+`' sera donc :
'`point operator+(point)`'.

Exemple

```
class point
{
       int x,y;
     public :
       point(int a=0,int b=0) {x=a; y=b;}
       void affiche()
       { cout<<"Point : "<<x<<" - "<<y<<endl; }
       point operator + (point);
};

//-------------------------------------------------

point point::operator+(point  p1)
{
     point p;
     p.x=x+p1.x; p.y=y+p1.y;
     return p;
}

//----------------------------

main()
{
     point     o1(10,20);     o1.affiche();
     point     o2(40,50);     o2.affiche();
     point     o3;            o3.affiche();
     o3=o1+o2;                o3.affiche();
     o3=o3+o1+o2;             o3.affiche();
}
```

Exécution



2. Les possibilités et les limites de la surdéfinition des opérateurs en C++

2.1. Il faut se limiter aux opérateurs existants

Le symbole suivant le mot clé 'operator' doit obligatoirement être un opérateur déjà défini pour les types de base, sauf l'opérateur point '.'. Il n'est donc pas possible de créer de nouveaux symboles.

Il faut conserver la pluralité (unaire, binaire) de l'opérateur initial.

Lorsque plusieurs opérateurs sont combinés au sein d'une même expression, ils conservent leurs priorités relatives et leurs associativités.

2.2. Tableau d'opérateurs surdéfinissabes, classés par priorité décroissante

Pluralité	Opérateur	Associativité
Binaire	( )◊ [ ] ◊ →◊	→
Unaire	+ - ++ -- ! & new◊ delete◊	←
Binaire	* / %	→
Binaire	+ -	→
Binaire	<< >>	→
Binaire	< <= > >=	→
Binaire	== !=	→
Binaire	& (niveau bit)	→
Binaire	^ (ou exclusif)	→
Binaire	||	→
Binaire	&&	→
Binaire	| (niveau bit)	→
Binaire	=◊ += -= *= /= %= &= ^= |= <<= >>=	←
Binaire	,	→

◊ : opérateur devant être surdéfini en tant que fonction membre.

2.3. Choix entre fonction membre et fonction amie

Si un opérateur doit absolument recevoir un type de base en premier argument, il ne peut pas être défini comme fonction membre (laquelle reçoit implicitement un premier argument du type de sa classe).

3. Exemple de surdéfinition de l'opérateur '[ ]'

Surdéfinir l'opérateur '[ ]' de manière à ce que 'o[i]' désigne l'élément du tableau dynamique d'emplacement 'i' de l'objet 'o' de la class '`tableau`'. Le premier opérande de 'o[i]' étant 'o'.

```
class tableau
{
        int ne;
        int *p;
      public :
        tableau(int n)
        {
          p=new int[ne=n]; for(int i=0;i<ne;i++) p[i]=(i+1)*10;
        }
        void affiche()
        {for(int i=0;i<ne;i++) cout<<p[i]<<"\t"; cout<<endl;}
        int operator[](int n){ return p[n]; }
        ~tableau(){delete []p;}
};
//---------------------------------------
main()
{
      tableau t1(3);
      t1.affiche();
      cout<<t1[2];
      t1.affiche();
}
```

La seule précaution à prendre consiste à faire en sorte que cette notation puisse être utilisée non seulement dans une expression, mais également à gauche d'une affectation.

Il est donc nécessaire que la valeur de retour fournie par l'opérateur '[ ]' soit transmise par référence :

```
class tableau
{
        int    ne;
        int     *p;
      public :
        tableau(int n)
        {
          p=new int[ne=n]; for(int i=0;i<ne;i++) p[i]=(i+1)*10;
        }
        void affiche()
        {for(int i=0;i<ne;i++) cout<<p[i]<<"\t"; cout<<endl;}
        int & operator[](int n){ return p[n]; }
        ~tableau(){delete []p;}
};
```

```
main()
{
       tableau t1(3);
       t1.affiche();
       cout<<t1[2];
       cout<<endl;
       t1[0]=55; cout<<t1[0];
       cout<<endl;
       t1.affiche();
}
```

Exécution

Remarque

L'opérateur '[]' n'est pas imposé ici par C++, on aurait pu le remplacer par l'opérateur '()' : 'o(i)' au lieu de : 'o[i]', ou un autre opérateur.

4. Exemple de surdéfinition de l'opérateur '='

Reprenons le cas de la classe 'tableau' :

```
class tableau
{
        int ne;
        int *p;
     public :
        ............;
};
```

Le problème de l'affectation est donc voisin de celui de la construction par recopie, mais non identique :

- Dans le cas de la construction par recopie (`tableau t1(3); tableau t2=t1;`), on a un seul tableau dynamique de 3 entiers pour les deux objets `t1` et `t2`.

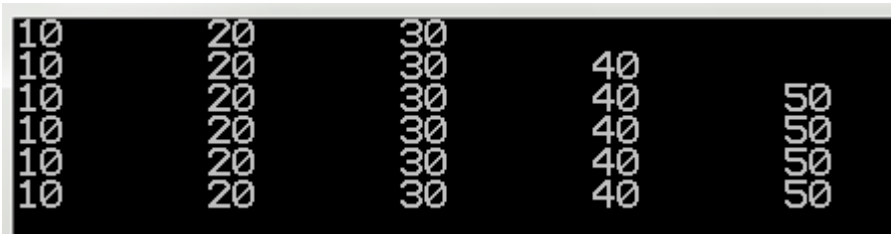
- Dans le cas d'affectation d'objets, il existe deux objets complets (avec leurs tableaux dynamiques). Mais après affectation (`t2=t1`), `t2` et `t1` référencent le même tableau dynamique (celui de `t1`), le tableau dynamique de `t2` n'est plus référencé.

- Ce problème peut être résolu en surdéfinissant l'opérateur d'affectation ; de manière à ce que chaque objet de type '`tableau`' possède son propre emplacement dynamique.
- Dans ce cas, on est sûr qu'il n'est référencé qu'une seule fois, et son éventuel libération peut se faire sans problèmes.

- Une affectation `t2=t1` ; pourrait être traitée de la façon suivante :
  - Libération de l'emplacement pointé par le pointeur `p` de `t2`.
  - Création dynamique d'un nouvel emplacement dans lequel on recopie les valeurs de l'emplacement pointé par `t1`.
  - Mise en place des valeurs des membres données de `t2`.
  - Il faut décider de la valeur de retour fournie par l'opérateur d'affectation en fonction de l'utilisation que l'on souhaite faire (void ou autre).
  - Dans l'affectation `t2=t1`; `t2` est le premier opérande (ici this car l'opérateur '=' est une fonction membre) et `t1` devient le second opérande (ici t).

```
class tableau
{
        int    ne;
        int     *p;
      public :
        tableau(int n)
        {
          p=new int[ne=n];
          for(int i=0;i<ne;i++) p[i]=(i+1)*10;
        }
        void affiche(){for(int i=0;i<ne;i++) cout<<p[i]<<"\t";
                       cout<<endl;
                      }
        tableau & operator=(tableau & t)
        {
          delete []p;
          p=new int[ne=t.ne];
          for(int i=0;i<ne;i++) p[i]=t.p[i];
          return *this;
        }
        ~tableau()
        {
           delete []p;
        }
};
//---------------------------------------
main()
{
      tableau t1(3); t1.affiche();
      tableau t2(4); t2.affiche();
      tableau t3(5); t3.affiche();
      t1=t3;
      t1.affiche();  t3.affiche();
      t1=t2=t3;
      t1.affiche();
}
```

Exécution




```
class tableau
{
        int    ne;
        int     *p;
      public :
        tableau(int n)
        {
          p=new int[ne=n];
              for(int i=0;i<ne;i++) p[i]=(i+1)*10;
        }
        void affiche(){for(int i=0;i<ne;i++) cout<<p[i]<<"\t";
                       cout<<endl;}
        tableau & operator=(tableau & t)
        {
               if(this!=&t)
               {    delete []p;
                    p=new int[ne=t.ne];
                    for(int i=0;i<ne;i++) p[i]=t.p[i];
               }
               return *this;
         }
         ~tableau()
         {
               delete []p;
         }
};
//---------------------------------------
main()
{
      tableau t1(3); t1.affiche();
      t1=t1;
      t1.affiche();
}
```

Exécution

Exercice

Refaire le même traitement pour des tableaux dynamiques de caractères.