Chapitre 3. Classes et objets

1. Rappels

1.1.Concept objet

La programmation orienté objet permet d'améliorer la maintenabilité (La robustesse, La modularité, Lisibilité).

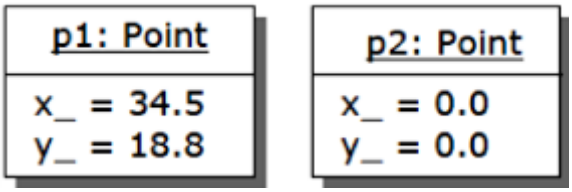
Grace à la programmation orienté objets, on peut organiser des programmes complexes en utilisant les notions suivantes : l'encapsulation, l'abstraction, l'héritage et de polymorphisme.

Un objet représente une entité individuelle et identifiable, réelle ou abstraite, avec un rôle bien défini, chaque objet peut être caractérisé par une identité, des états significatifs et par un comportement.

Objet = Etat + Comportement + Identité

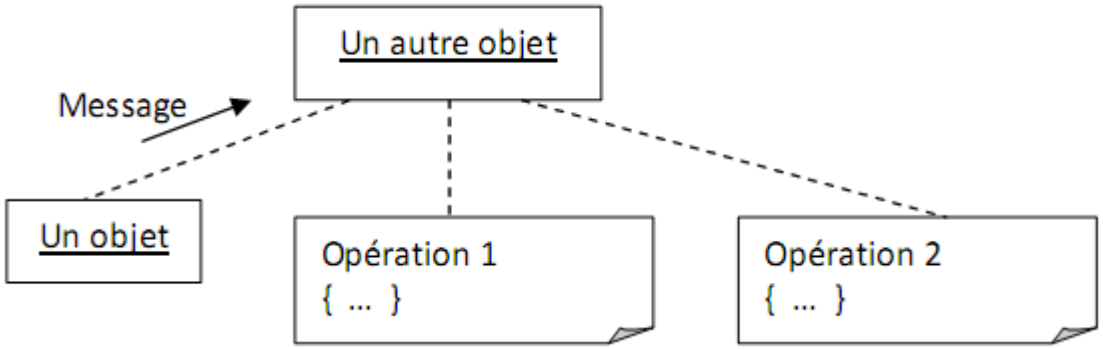
Etat : L'état d'un objet comprend les propriétés statiques (attributs) et les valeurs de ces attributs qui peuvent être statiques ou dynamiques.

Exemple : Les attributs de l'objet point en deux dimensions sont les coordonnées x et y



Comportement : Le comportement d'un objet se défini par l'ensemble des opérations qu'il peut exécuter en réaction aux messages envoyés (un message = demande d'exécution d'une opération) par les autres objets.

Exemple: Un point peut être déplacé, tourné autour d'un autre point, etc.



Identité : Propriété d'un objet qui permet de le distinguer des autres objets de la même classe.

2. Notion de classe

Si des objets ont les mêmes attributs et comportements: ils sont regroupés dans une famille appelée: **Classe**. Donc, des objets similaires peuvent être informatiquement décrits par une même abstraction: une classe (Les objets appartenant à celle-ci sont les instances de cette classe) :

- Même structure de données et méthodes de traitement ;
- Valeurs différentes pour chaque objet.

3. Déclaration

Une classe contient donc :

- des données-membres ou attributs.
- des fonctions-membres ou méthodes.

Une classe se définit par le mot clef "**class**" suivi du nom de celle-ci (généralement commencé par une majuscule) et les données membres et les méthodes de la classe sont déclarées entre

deux accolades (n'oubliez pas que la définition de la classe se termine obligatoirement par un point-virgule). Le squelette d'une classe est donné par le synatxe suivant :

Syntaxe:

```
class Nom_class
{
private:
     // Déclaration des attributs
    // et méthodes privés
public:
   // Déclaration des attributs
   // et méthodes publics
         };
```

Exemple

L'exemple suivant permet de définir la classe Point aux coordonnées x et y (les attributs de classe) et les fonctions membres **afficher, changer et ajouter** (les méthodes).

Le terme public signifie que tous les membres qui suivent (données comme méthodes) sont accessibles de l'extérieur de la classe.

```
#include <iostream>
using namespace std;

class Point
{  public : // voici les attributs
    int x;
    int y;
   // voici les méthodes
   void afficher()
   { cout <<x<<','<<y<<endl; }
 void changer (int a ,int b)
   {  x=a;
      y=b;
   }
   void ajouter(int a, int b)
   {  x += a;
      y += b;
   }
};
```

..

il est possible de créer des objets (des variables ou des constantes) de ce type. Pour déclarer un "objet" d'un type de classe donné, il suffit de précéder son nom de celui de la classe. Différents objets d'une même classe disposent des mêmes attributs et des mêmes méthodes, mais les valeurs des attributs sont différentes pour chaque objet.

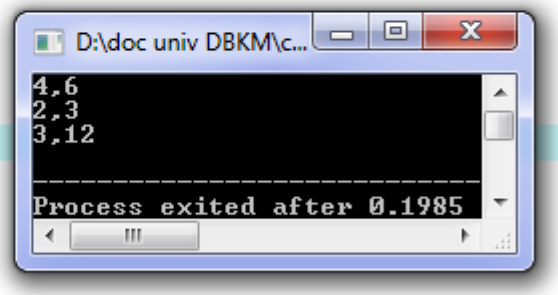
Exemple :

```
Point A,B;// on declare deux objets A et B  de type point
```

4. Accès au membre d'un objet

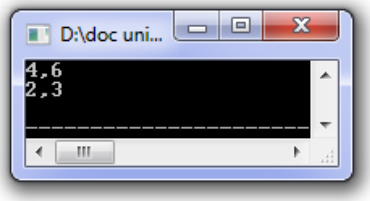
4.1.Déclaration statique

```
#include <iostream>
using namespace std;
class Point
{  public : // voici les attributs
    int x;
    int y;
   // voici les méthodes
   void afficher()
   { cout <<x<<','<<y<<endl; }
 void changer (int a ,int b)
   {  x=a;
      y=b;
   }
   void ajouter(int a, int b)
   {  x += a;
      y += b;
   }
};
int main()
{
  Point p;
  p.x=4;
  p.y=6;
  p.afficher();
  p.changer(2,3);
  p.afficher();
  p.ajouter(1,9);
   p.afficher();
}
```



Ou bien

```
// déclaration static
//=================================================
///*
#include <iostream>
#include "Point.cpp"
int main()
{
  Point p;
  p.x=4;
  p.y=6;
  p.afficher();
  p.changer(2,3);
  p.afficher();
  return 0;
}
```

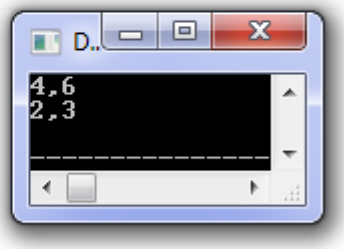


[*] main.cpp | Point.cpp

```
1   using namespace std;
2   class Point
3   {  public : // voici les attributs
4       int x;
5       int y;
6      // voici les méthodes
7      void afficher()
8      { cout <<x<<','<<y<<endl; }
9    void changer (int a ,int b)
10     {  x=a;
11        y=b;
12     }
13     void ajouter(int a, int b)
14     {  x += a;
15        y += b;
16     }
17  };
```

4.2.Allocation Dynamique

```
//=========================================
// Allocation dynamique
//=========================================
//*
 #include <iostream>
#include "Point.cpp"
int main()
{
Point *p=new Point;
  p->x=4;
  p->y=6;
  p->afficher();
  p->changer(2,3);
  p->afficher();
  return 0;
}
```

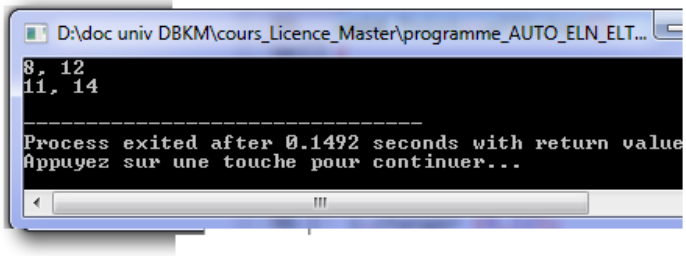


5. Opérateurs de résolution de portée

Les méthodes de la classe Point sont implémentées dans la classe elle-même. Lorsque le code est plus long, cela devient assez lourd. Il est donc préférable de placer la déclaration uniquement, dans la classe.

Exemple

```
//Opérateurs de résolution de portée
//=========================================
#include <iostream>
using namespace std;
class Point
{ public :
   int x;
   int y;
   void changer(int a, int b);
   void ajouter(int a, int b);
   void afficher();
};
void Point::changer(int a, int b)
{    x = a;
     y = b;
}
void Point::ajouter(int a, int b)
{  x += a;
   y += b;
}
void Point::afficher()
{
   cout << x << ", " << y << endl;
}
int main()
{  Point z;

 z.changer (8,12);
 z.afficher();
z.ajouter(3,2);
   z.afficher();
   return 0;
}
```



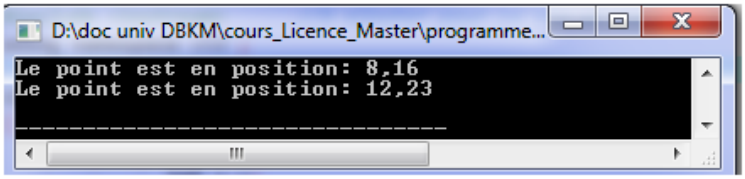
"Point ::" (:: opérateur de résolution de portée) signifie que la fonction est une méthode de la classe Point.

6. Conception d'un programme Orienté Objet

Pour illustrer la notion d'objet et de classe, nous utilisons l'exemple simple suivant :

Nous réalisons une classe point permettant de manipuler un point dans un plan. Cette classe comporte deux membres données privées : x et y et trois fonctions membres (ou méthodes) publiques : **initialiser, deplacer, afficher.**
D'abord, on déclare la classe au début du programme après le mot clé class, puis on définit le contenu des méthodes membres.
obj1 est un objet de classe point.

```
//Conception d'un programme Orienté Objet: exemple 1
//=============================================================================
#include <iostream>      //à ajouter pour le cin et cout
using namespace std ;
 /*création de la classe point*/
class point
{
      private :
            int x,y;
      public :
            void initialiser(int a,int b);
            void changer(int dx,int dy);
            void afficher();
};
void point::initialiser(int a, int b)
{    x=a;
     y=b;
}
void point::changer(int dx,int dy)
{    x=x+dx;
     y=y+dy;
}
 void point::afficher()
{  cout<<"Le point est en position: "<<x<<","<<y<<endl;
}
int main()
{     point obj1; //construction d'un point obj1
      obj1.initialiser(8,16);//initalisation de ce point de  coordonnées x=8, y=16
      obj1.afficher();//affichage de ce point
      obj1.changer(4,7);//déplacement de ce point à droite
      obj1.afficher();//nouvel affichage
     return 0 ;
}
```



7. Déclaration et instanciation

Tout objet est instance d'une classe. Dans la classe point, on remarque que la création de l'objet obj1 se fait en deux étapes :

- Déclaration de l'objet ;
- Instanciation de l'objet.

La déclaration crée une nouvelle variable. A cette étape, aucune réservation de mémoire n'a eu lieu pour cet objet. Il est donc inutilisable.
L'instanciation est l'opération qui consiste à créer un nouvel objet à partir d'une classe. Elle permet de réserver une zone mémoire spécifique pour l'objet. Sans cette étape, l'objet déclaré ne peut pas être utilisé.

7.1.Instanciation d'un objet

Pour instancier un objet : On doit le déclarer, pour le déclarer il faut donner le type de l'objet. Le type d'un objet est sa classe. Etant donné une instance d'un objet, on accède à ses attributs et à ses méthodes grâce à la notation pointée.
Dans l'exemple précédent sur la classe point. On a d'abord déclaré un objet appelée obj1 de type point. Puis on l'a initialisé à la position (x,y)=(8,16).

```
point obj1; //construction d'un point obj1
obj1.initialiser(8,16);//initalisation de ce point de  coordonnées x=8, y=16
```

7.2.Instanciation de plusieurs objets

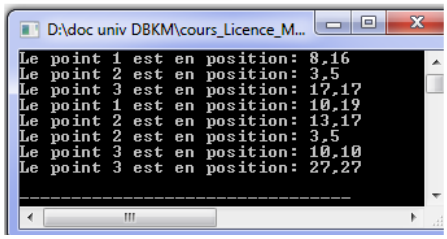
On peut instancier plusieurs d'objet d'une même classe Une fois les objets déclarés

Exemple 2

En utilisant le code fourni ci-dessus sur la conception de la classe point, pour instancier plusieurs objets de type point :

1. Créez un objet obj2 de type point avec les valeurs suivantes : (3,5).
2. Faites un déplacement de obj2 à droite de (10,12).
3. Ajoutez une méthode changerG qui déplace le point à gauche.
4. Faites un déplacement de obj2 à gauche de (10,12).
5. Créez un objet obj3 de type point avec les valeurs suivantes : (17,17).
6. Faites un déplacement de obj3 à gauche de (7,7) et à droite de (17,17).

```
//===================================
#include <iostream>
using namespace std ;
 //création de la classe point
class point
{
private :
    int x,y;
public :
     void initialiser(int a,int b);
     void changerD(int dx,int dy);
     void changerG(int dx,int dy);
     void afficher(int n);
};
void point::initialiser(int a, int b)
{
    x=a;
    y=b;
}
void point::changerD(int dx,int dy)
{
    x=x+dx;
    y=y+dy;
}
void point::changerG(int dx,int dy)
{
    x=x-dx;
    y=y-dy;
}
void point::afficher(int n)
{
    cout<<"Le point "<<n<<" est en position: "<<x<<","<<y<<endl;
}
int main()
{
    point obj1,obj2,obj3;         //déclaration d'un point obj1
    obj1.initialiser(8,16); //initalisation de ce point de coordonnées x=8, y=16
    obj1.afficher(1);

    obj2.initialiser(3,5);
    obj2.afficher(2);           //affichage du point obj2

    obj3.initialiser(17,17);
    obj3.afficher(3);          //affichage du point obj3

    obj1.changerD(2,3); //déplacement de obj1 à droite
    obj1.afficher(1);

    obj2.changerD(10,12);
    obj2.afficher(2);  //nouvel affichage de obj2

    obj2.changerG(10,12);
    obj2.afficher(2);

    obj3.changerG(7,7);
    obj3.afficher(3);

    obj3.changerD(17,17);
    obj3.afficher(3);
    return 0;
}
```



Exemple 3

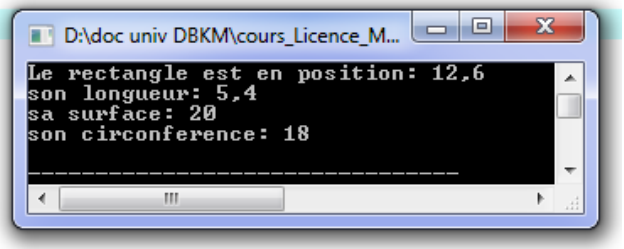
Il s'agit ici de définir une classe rectangle représentant une abstraction (informatique) de ce qu'est un rectangle : une largeur, une longueur, sa surface et son circonférence. Cette classe

contient un emplacement x et y une **longueur** et une **largeur** comme variables, les fonctions **initialiser, deplacer, surface, circonference** et **afficher** sont leurs méthodes.

- La fonction **initialiser** permet d'initialiser la largeur et longueur x, y.
- La fonction **deplacer** permet de déplace le rectangle par un décalage à droite de dx et dy.
- La fonction surface permet de renvoyer la valeur la surface du rectangle.
- La fonction **circonference** permet de renvoyer la valeur de la circonférence du rectangle.
- La fonction **afficher,** permet d'affiche la position, la longueur, la largeur, la surface ainsi que le périmètre du rectangle.

D'abord on doit définir une classe rectangle permettant de manipuler les fonctions définies ci-dessus. Utilisez cette classe pour instancier un objet rec1 :
a) Crée un objet rec1 de type rectangle.
b) Initialise le rec1 par les valeurs (10,2,5,4).
c) Fait un déplacement de rec1 de (2,4).
d) Calcule la surface et le périmètre de rec1.
e) Affiche les résultats.

```
//=========================================================================
#include <iostream>
using namespace std;
 class rectangle
{
     private :
          int x,y;
          float lon,lar;
     public :
          void initialiser(int a,int b, int c, int d);
          void deplacer(int dx, int dy);
          float surface();
          float circonference();
          void afficher();
};
void rectangle::initialiser(int a, int b, int c, int d)
{
     x=a;
     y=b;
     lon=c;
     lar=d;
}
```



```
void rectangle::deplacer(int dx,int dy)
{
    x=x+dx;
    y=y+dy;
}
float rectangle::surface()
{
    return lon*lar;
}
float rectangle::circonference()
{
    return 2*(lon+lar);
}
void rectangle::afficher()
{
    cout<<"Le rectangle est en position: "<<x<<","<<y<<"\n"
    <<"son longueur: "<<lon<<","<<lar<<"\n"
    <<"sa surface: "<<surface()<<"\n"
    <<"son circonference: "<<circonference()<<endl;
}
    <<"sa surface: "<<surface()<<"\n"
    <<"son circonference: "<<circonference()<<endl;
}
int main()
{
    rectangle rec1;
    rec1.initialiser(10,2,5,4);
    rec1.deplacer(2,4);
    rec1.surface();
    rec1.circonference();
    rec1.afficher();
       return 0;
}
```

8. Constructeur, Destructeur

Toute classe nécessite d'être crée puis détruite, pour cela il faut un constructeur et un destructeur. Le constructeur alloue implicitement de la mémoire et permet d'initialiser les variables internes. Le destructeur quant à lui est appelé lorsque la classe va être détruite et libère ainsi la mémoire.

8.1.Les Constructeurs

Tout constructeur doit répondre aux caractéristiques suivantes :

-Est une fonction membre qui porte le même nom que sa classe,

-Est appelé après l'allocation de l'espace mémoire destiné à l'objet,

-Ne renvoie pas de valeur (pas même void ne doit figurer devant sa déclaration ou sa définition).

-Si aucun constructeur n'est déclaré, un constructeur par défaut, sans paramètres, est automatiquement créé.

Il existe trois types de constructeurs :

1-Constructeur par défaut (sans paramètres)

2-Constructeur d'initialisation ou paramétré (avec paramètres)

3-Constructeur par copie (reçoit en paramètre un objet)

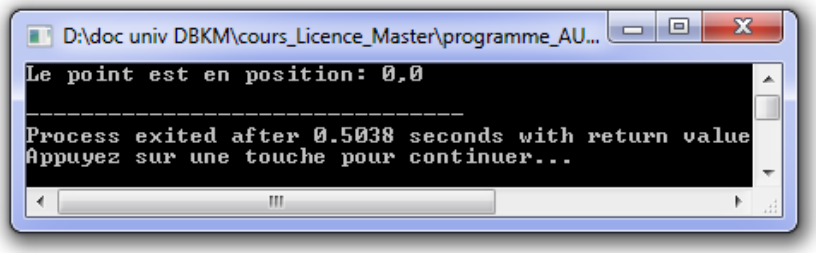
8.1.1. Constructeur par défaut (le constructeur sans paramètres).

Une classe qui ne déclare aucun constructeur explicitement en possède en fait toujours un : le constructeur vide par défaut, qui ne prend aucun paramètre, il définira les variables à 0 et les chaînes à vide.

Exemple

Un constructeur vide par défaut est créé dans cette classe

```
#include <iostream>
using namespace std;
class point
{      private :
            int x,y;
       public :
            point();
            void affiche();
};
point::point()
{
     x=0;
     y=0;
}
void point::affiche()
{
     cout<<"Le point est en position: "<<x<<","<<y<<endl;}
    int main()
{
  point p1 ;
p1.affiche();
}
```

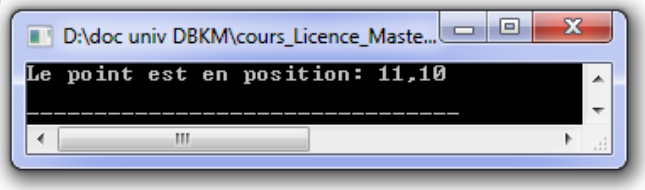


8.1.2. Constructeur d'initialisation (le constructeur paramétré)

Si l'on définit un constructeur explicite dans cette classe (i.e. la classe point), alors ce constructeur vide par défaut n'existe plus (il n'est plus créé) ; on doit le déclarer explicitement si l'on veut encore l'utiliser. On ne peut plus instancier cette classe comme précédemment. On ne peut l'instancier que de la façon suivante :

Exemple

```
#include <iostream>
using namespace std;
class point
{      private :
            int x,y;
       public :
            point(int a, int b);
            void affiche();
};
point::point(int a, int b)
{
     x=a;
     y=b;
}
void point::affiche()
{
     cout<<"Le point est en position: "<<x<<","<<y<<endl;}
 int main()
 {
    point p1(11,10) ;
 p1.affiche();
     }
```

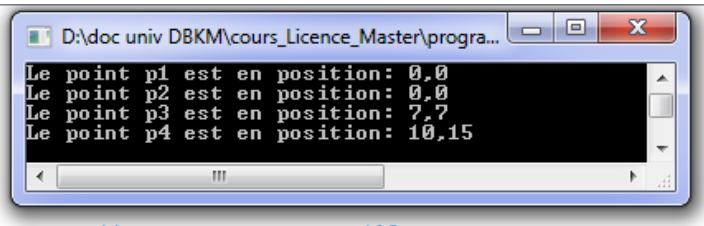


8.1.3. Surdéfinition ou surcharge de constructeurs

Les constructeurs peuvent être surchargés si la classe contient plus d'un constructeur.

Exemple

```
#include <iostream>
using namespace std;
class point
{
private :
    int x,y;
public :
      point();                    // constructeur par défaut
      point(int a);               // constructeur d'initialisation avec un seul paramétre
      point(int a, int b);        // constructeur d'initialisation avec deux paramétres
      void affiche(int n);
};
point::point()  //initialise x à la valeur 0 et y à la valeur 0
{
     x=0;
     y=0;
}
point:: point(int a) //initialise x à la valeur a et y à la valeur a
{
     x=a;
     y=a;
}
void point::affiche(int n)
{
     cout<<"Le point p"<<n<<" est en position: "<<x<<","<<y<<endl;
}
int main()
 {
   point p1,p2;// appel du constructeur par défaut par les objets p1 et p2
   point p3(7) ;// appel du constructeur d'initialisation avec un seul paramétre par l'objet p3
   point p4(10,15); // appel du constructeur d'initialisation avec deux paramétres par l'objet p4
   p1.affiche(1);
   p2.affiche(2);
   p3.affiche(3);
   p4.affiche(4);
}
```



On peut utiliser un tableau d'une dimension. L'instanciation devient alors :

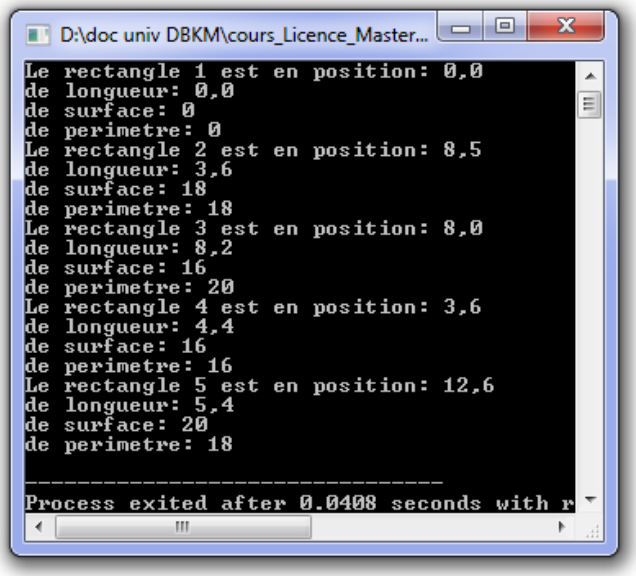
```
   point p[3] = {point(7),point(10,15)} ;    // appel du constructeur par défaut par les deux
                                              //premiers éléments du tableau et un constructeur
                                              // d'initialisation avec un seul paramétre pour le 3éme élément
                                              // d'initialisation avec deux paramétres pour le 4éme élément
for (int i=0 ;i<4 ;i++)
{p[i].affiche(i);}
```

Exemple 2

Utilisez la classe rectangle de l'exemple précédent., on utilisera cinq (5) constructeurs, qui sont définis comme suit :

1. Un constructeur par défaut : rectangle();
2. Un constructeur d'initialisation ou paramétré avec la longueur donnée : **rectangle(lon1)** ;
3. Un constructeur d'initialisation ou paramétré avec la largeur donnée : **rectangle(lar1)** ;
4. Un constructeur d'initialisation avec deux paramètres qui sont la longueur et la largeur : **rectangle(lon1, lar1) ;**
5. Un constructeur d'initialisation avec les position x et y initialisé ainsi que les dimensions x et y : **rectangle(x1,y1,lon1, lar1);**

```
//
//Surdéfinition ou surcharge de constructeurs  exemple 2
//=============================================================================================
#include <iostream>
using namespace std;

class rectangle
{
     private :
          int x,y;
          float lon,lar;
     public :
           rectangle();
           rectangle(int);
           rectangle(int,int);
           rectangle(int,int, int);
           rectangle(int,int, int, int);
           void deplace(int dx, int dy);
           int surface();
           int circonference();
           void affiche(int n);
};
 rectangle::rectangle()
{
     x=0;
     y=0;
    lon=0;
    lar=0;
}
 rectangle::rectangle(int lon1)
{
     x=8;
     y=5;
     lon=lon1;
     lar=6;
}
 rectangle::rectangle(int lon1, int lar1)
{
     x=8;
     y=0;
     lon=lon1;
     lar=lar1;
}
 rectangle::rectangle(int x1, int y1,int lar1)
{
      x=x1;
      y=y1;
     lon=lar1;
     lar=lar1;
}
 rectangle::rectangle(int x1, int y1, int lon1, int lar1)
{
     x=x1;
     y=y1;
     lon=lon1;
```



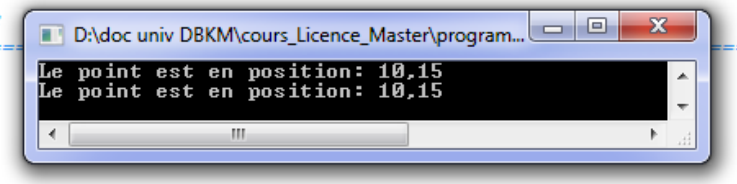
```
    lar=lar1;
}
void rectangle::deplace(int dx,int dy)
{
     x=x+dx;
     y=y+dy;}
    int rectangle::surface()
{
     return lon*lar;
}
int rectangle::circonference()
{
     return 2*(lon+lar);
}
void rectangle::affiche(int n)
{
     cout<<"Le rectangle "<<n<<" est en position: "<<x<<","<<y<<"\n"
     <<"de longueur: "<<lon<<","<<lar<<"\n"
     <<"de surface: "<<surface()<<"\n"
     <<"de perimetre: "<<circonference()<<endl;
}
int main()
{
     rectangle rec1;
     rectangle rec2(3);
     rectangle rec3(8,2);
     rectangle rec4(3,6,4);
     rectangle rec5(10,2,5,4);
        rec1.surface();
     rec1.surface();
     rec1.circonference();
    rec1.affiche(1);
       rec2.surface();
    rec2.circonference();
    rec2.affiche(2);
       rec3.surface();
    rec3.circonference();
    rec3.affiche(3);
     rec4.surface();
    rec4.circonference();
    rec4.affiche(4);
      rec5.deplace(2,4);
    rec5.surface();
    rec5.circonference();
    rec5.affiche(5);

    return 0;
}
```

8.1.4. Constructeur de copie

Il doit effectuer une copie d'un objet du même type. Il sert à créer une copie explicitement,
Il est aussi utilisé implicitement par le compilateur, comme le constructeur sans arguments, toutes les classes en ont un par défaut.
Ce constructeur par défaut recopie simplement toutes les données membres de l'objet initial transmises en arguments comme données membres du nouvel objet.

Exemple :

La classe point définit un constructeur de copie qui prend comme argument une instance de point. Les valeurs des propriétés de l'argument sont assignées aux propriétés de la nouvelle instance de point. Le code contient un autre constructeur de copie qui envoie les propriétés x et y de l'instance que vous voulez copier au constructeur d'instance de la classe.




```
//=============================================================================================
// Constructeur de copie
//=============================================================================================
#include <iostream>
using namespace std;
class point
{
public :
      point(int a, int b);  // constructeur d'initialisation
void affiche();
     int x,y;
};
point:: point(int a, int b) //initialise x à la valeur a et y à la valeur b
{     x=a;
y=b;}
void point::affiche()
{cout<<"Le point est en position: "<<x<<","<<y<<endl;}

//L'instanciation de l'objet p2 se fait par un constructeur de copie de la façon suivante :

int main()
{
point p1(10,15) ;// création d'un objet p1 par le constructeur d'initialisation
point p2=point(p1) ;// création d'un objet p2 par le constructeur de copie
//affichage des résultats
p1.affiche();
p2.affiche();
}
```

On remarque que les coordonnées du point x et y ne sont plus private, puisqu'on les modifie après les avoir construit.

8.2.Le destructeur

Le destructeur permet de détruire l'objet lorsqu'il devient inutile. Les caractéristiques d'un destructeur sont :

-Il est unique.

-est une fonction membre portant le même nom que sa classe, précédé du symbole (~),

-est appelé avant la libération de l'espace mémoire associé à l'objet

-ne peut pas comporter d'arguments et il ne renvoie pas de valeur (aucune indication de type ne doit être prévue).

- Il doit restituer la place mémoire allouée dynamiquement par l'objet.

Le destructeur est une méthode particulière qui est définie implicitement pour tous les objets. Par défaut il ne fait rien

Exemple 1 :

```
//=============================================================================
#include <iostream>
#include <conio.h>
using namespace std ;



class rectangle
{
     private :
          int x,y;
          float lon,lar;
          static int ctr ; // compteur d'objets
     public :
           rectangle (int,int,float,float); // Constructeur
           float surface();
           void affiche(int);
           ~rectangle(); // Destructeur
};
 int rectangle::ctr = 0; // init. A 0 du nb d'objets rectangle

rectangle::rectangle(int a, int b, float c, float d) //constructeur
{
        x=a;
        y=b;
        lon=c;
        lar=d;
       ctr++;
}

float rectangle::surface()
{
     return lon*lar;
}

void rectangle::affiche(int n)
{
      cout<<"Le rectangle "<<n<<" est en position: "<<x<<","<<y<<"\n"
     <<"de longueur: "<<lon<<","<<lar<<"\n"
     <<"de surface: "<<surface()<<endl;
   }
rectangle::~rectangle() // destructeur
{
     ctr-- ;
     cout << "Nombre d'objets rect restants : " << ctr << endl ;
}

int main()
{
     rectangle rec1(10,2,5,4);   //construction d'un rectangle rec1
     rectangle rec2(20,4,10,8); //construction d'un rectangle rec2
     rec1.surface();
     rec2.surface();
    rec1.affiche(1);
    rec2.affiche(2);

     rec2.~rectangle();        //destruction du rectangle rec2
    getch();
       return 0;
}
```

```
D:\doc univ DBKM\cours_Licence_Master\programme_AUTO_E...
Le rectangle 1 est en position: 10,2
de longueur: 5,4
de surface: 20
Le rectangle 2 est en position: 20,4
de longueur: 10,8
de surface: 80
Nombre d'objets rect restants : 1
```

9. L'encapsulation

En P.O.O, l'encapsulation signifie que leurs accès ne peuvent se faire que par le biais des méthodes. La déclaration d'une classe précise quels sont les membres (données ou fonctions)

publics (c'est-à-dire accessibles à l'utilisateur de classe), on utilise dans ce cas le mot public et quels sont les membres privés (inaccessibles par l'utilisateur de la classe), on utilise dans ce cas le mot private.

9.1.Différents niveaux d'accessibilité des propriétés et méthodes

Il existe trois catégories définissant les restrictions d'accès aux fonctions et variables, ceci renforce l'encapsulation ou le masquage des données. Ces catégories sont définies par les mots-clés public, private ou protected.

9.1.1. Private

Une fonction private ne peut être appelée qu'à partir des fonctions membres de la classe, et qu'une donnée membre private ne peut être lue ou modifiée que par des fonctions membres de la classe.

9.1.2. Public

Pour les données et variables membres public, il n'y a pas de restriction et on peut y accéder à l'extérieur comme à l'intérieur de l'objet.

9.1.3. Protected

Il s'agit d'un cas intermédiaire entre private et public. Les fonctions et variables membres de type protected sont accessibles par toute fonction membre de l'objet ou d'une des classes dérivées de l'objet (La notion de classe dérivée se reportera au prochaine chapitre traitant des notions d'héritage et de polymorphisme).