Programmation C++ (débutant)/Les classes

Le cours du chapitre 11 : Les classes

Une évolution des structures

Une fois introduite la notion de structures, on s'aperçoit qu'il est totalement naturel de créer des fonctions permettant de manipuler ces structures. La notion de classe est donc une notion plus puissante que la notion de structures. Une classe va permettre de regrouper en une seule entité des données membres et des fonctions membres appelées méthodes.

Cependant, contrairement au langage C, les structures du C++ permettent cela également. La différence entre une structure et une classe est que les attributs et méthodes d'une structure sont par défaut publics, alors qu'ils sont par défaut privés dans une classe. Une autre différence est que les structures ne peuvent utiliser l'héritage.

Notion de classe

Une classe regroupera donc :

- des données membres.
- des méthodes membres qui seront des fonctions.

Un premier exemple de classe

- On veut manipuler des points définis par une abscisse et une ordonnée (des réels).
- Sur un point, on peut calculer la distance entre 2 points et le milieu de 2 points.
- Nous allons donc définir une classe Point définie par un fichier .h et un fichier .cpp.

Exemple 1 : la classe Point

Le fichier Point.h

```
#ifndef POINT_H
#define POINT_H

class Point
{
  public:
    double x,y;
    double distance(const Point &P);
    Point milieu(const Point &P);
};
#endif
```

Explications

On définit dans ce fichier la classe Point : elle contient 2 données de type double x et y et 2 méthodes membres distance qui calcule la distance entre ce point et un autre Point et milieu qui calcule le milieu du segment composé de ce point et d'un autre Point.

On remarque l'utilisation des directives de compilation #ifndef, #define et #endif pour gérer les inclusions multiple du fichier header.

Le fichier Point.cpp

```
#include "Point.h"
#include <cmath>

double Point::distance(const Point &P)
{
    double dx,dy;
    dx = x - P.x;
    dy = y - P.y;
    return sqrt(dx*dx + dy*dy);
}

Point Point::milieu(const Point &P)
{
    Point M;
    M.x = (P.x+x) /2;
    M.y = (P.y+y) /2;
    return M;
}
```

Explications

- Il contient l'implémentation de chaque méthode de la classe Point.
- On fait précéder chaque méthode de Point::
- On a inclut le fichier cmath afin de pouvoir utiliser la fonction sqrt de cmath (racine carrée).
- A l'intérieur de la classe Point, on peut accéder directement à l'abscisse du point en utilisant la donnée membre x.
- On peut accéder à l'abscisse du paramètre P d'une méthode en utilisant P.x.

Le fichier main.cpp

```
#include <iostream>
using namespace std;
#include "Point.h"

int main()
{
    Point A, B, C;
    double d;
    cout << "SAISIE DU POINT A" << endl;
    cout << "Tapez l'abscisse : "; cin >> A.x;
    cout << "Tapez l'ordonnée : "; cin >> A.y;
    cout << endl;
```

```
    cout << "SAISIE DU POINT B" << endl;
    cout << "Tapez l'abscisse : "; cin >> B.x;
    cout << "Tapez l'ordonnée : "; cin >> B.y;
    C = A.milieu(B);
    d = A.distance(B);
    cout << endl;
    cout << "MILIEU DE AB" << endl;
    cout << "L'abscisse vaut : " << C.x << endl;
    cout << "L'ordonnée vaut : " << C.y << endl;
    cout << endl;
    cout << "La distance AB vaut :" << d << endl;
    return 0;
}
```

Explications

- Une fois inclus le fichier d'en-tête Point.h, on peut définir 3 points A, B et C.
- A, B et C sont 3 objets qui sont des instances de la classe Point.
- Les données membres étant publiques, on peut accéder à l'abscisse et à l'ordonnée de A en dehors de la classe en écrivant A.x et A.y.
- Les méthodes membres distance et milieu étant publiques, on peut écrire directement A.milieu(B) ou A.distance(B).

Exécution

Lorsqu'on exécute ce programme, il s'affiche à l'écran :

```
SAISIE DU POINT A
Tapez l'abscisse : 3.2
Tapez l'ordonnée : 1.4
SAISIE DU POINT B
Tapez l'abscisse : 5
Tapez l'ordonnée : 6
MILIEU DE AB
L'abscisse vaut : 4.1
L'ordonnée vaut : 3.7
La distance AB vaut : 4.93964
```

Encapsulation

- Il faut éviter de donner un accès extérieur aux données membres d'un objet quelconque.
- On va interdire l'accès à certaines données membres d'une classe ou certaines méthodes en utilisant le mot clé `private`.
- On ne peut accéder à une variable (ou une méthode membre) privée que par l'intérieur de la classe.
- par contre, on peut accéder librement à toutes les données membres ou méthodes membres publiques.
- Cette technique fondamentale permet d'empêcher au programmeur de faire n'importe quoi : il ne pourra accéder à ces données que par les méthodes publiques.

Interface et boîte noire

- Vu de l'extérieur, on ne peut accéder à un objet donné que grâce à ces méthodes publiques.
- Ceci permet entre autre de protéger l'intégrité des données.
- L'ensemble des méthodes publiques est appelée l'interface de l'objet.
- De l'extérieur, l'objet peut être vu comme une boîte noire qui possède une interface d'accès.
- On cache ainsi à l'utilisateur de cette classe comment cette interface est implémentée : seul le comportement de l'interface est important.

Accesseurs et mutateurs

- On pourra accéder aux valeurs des données membres privées grâce à des méthodes spécifiques appelée accesseurs. Les accesseurs seront publics.
- On pourra même modifier ces valeurs grâce à des fonctions spécifiques appelées mutateurs.
- Cela permet au programmeur d'avoir un contrôle complet sur ces données et sur des contraintes en tout genre qu'il veut imposer à ces données.

Exemple 2 : accesseurs et mutateurs

Le fichier Point.h

```
#ifndef POINT_H
#define POINT_H

class Point
{
  public:
    void setX(double x);
    void setY(double y);
    double getX();
    double getY();
    double distance(const Point &P);
    Point milieu(const Point &P);
    void saisir();
    void afficher();

  private:
    double x,y;
};
#endif
```

Explications

- La méthode void setX(double x) est un mutateur qui permet de modifier la donnée membre privée x.
- Idem pour void setY(double y) avec la donnée membre privée y.
- Les méthodes double getX() et double getY() sont des accesseurs qui permettent d'accéder aux valeurs respectives des données membres privées x et y.
- Les méthodes saisir() et afficher() permettent respectivement de saisir et d'afficher les coordonnées des points.

Le fichier Point.cpp

```
#include "Point.h"
#include <cmath>
#include <iostream>
using namespace std;

void Point::setX(double x)
{
    this->x = x;
}
void Point::setY(double y)
{
    this->y = y;
}

double Point::getX()
{
    return x;
}

double Point::getY()
{
    return y;
}

double Point::distance(const Point &P)
{
    double dx, dy;
    dx = x - P.x;
    dy = y - P.y;
    return sqrt(dx*dx + dy*dy);
}

Point Point::milieu(const Point &P)
{
    Point M;
    M.x = (P.x + x) /2;
    M.y = (P.y + y) /2;
    return M;
}
```

```
void Point::saisir()
{
    cout << "Tapez l'abscisse : "; cin >> x;
    cout << "Tapez l'ordonnée : "; cin >> y;
}

void Point::afficher()
{
    cout << "L'abscisse vaut " << x << endl;
    cout << "L'abscisse vaut " << y << endl;
}
```

Explications

- Dans la méthode setX(...), il y a une utilisation du mot-clé this. Le mot clé this désigne un pointeur vers l'instance courante de la classe elle même. this->x désigne donc la donnée membre de la classe alors que x désigne le paramètre de la méthode void setX(double x);
- Le mutateur double getX(); se contente de renvoyer la valeur de x.

Le fichier main.cpp

```
#include <iostream>
using namespace std;
#include"Point.h"

int main()
{
    Point A, B, C;
    double d;
    cout << "SAISIE DU POINT A" << endl;
    A.saisir();
    cout << endl;
    cout << "SAISIE DU POINT B" << endl;
    B.saisir();
    cout << endl;
    C = A.milieu(B);
    d = A.distance(B);
    cout << "MILIEU DE AB" << endl;
    C.afficher();
    cout << endl;
    cout << "La distance AB vaut :" << d << endl;
    return 0;
}
```

Explications

- On n'a plus le droit d'accéder aux données membres x et y sur les instances de Point A, B et C en utilisant A.x ou B.y : il faut obligatoirement passer là une des méthodes publiques.
- Pour saisir la valeur de A, il suffit d'écrire A.saisir();
- Pour afficher la valeur de A, il suffit d'écrire A.afficher();

Exécution

Lorsqu'on exécute ce programme, il s'affiche à l'écran :

```
SAISIE DU POINT A
Tapez l'abscisse : 3.2
Tapez l'ordonnée : 1.4
SAISIE DU POINT B
Tapez l'abscisse : 5
Tapez l'ordonnée : 6
MILIEU DE AB
L'abscisse vaut : 4.1
L'ordonnée vaut : 3.7
La distance AB vaut : 4.93964
```

Utiliser les opérateurs >> et <<

- Pour pouvoir saisir un Point au clavier, on pourrait écrire tout simplement cin >> A; où A est une instance de la classe Point.
- Pour écrire un Point à l'écran, on peut écrire tout simplement cout << B.
- Nous allons utiliser pour cela les fonctions operator>> et operator<<.
- Dans l'exemple 3, on utilisera une manière de procéder assez personnelle sans utiliser de fonctions amies;
- Dans l'exemple 4, nous verrons la méthode qui semble plus classique basée sur les fonctions amies.

Exemple 3 : l'opérateur >> et l'opérateur <<

Fichier Point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>
using namespace std;

class Point
{
  public:
    void setX(double x);
    void setY(double y);
    double getX();
    double getY();
    double distance(const Point &P);
    Point milieu(const Point &P);
    void operator>>(ostream &out);
    void operator<<(istream &in);
```