Programmation C++ (d€butant)/Les classes

Le cours du chapitre 11 : Les classes

Une €volution des structures

Une fois introduite la notion de structures, on s'aper•oit qu'il est totalement naturel de cr€er 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 entit€ des donn€es membres et des fonctions membres appel€es m€thodes.

Cependant, contrairement au langage C, les structures du C++ permettent cela €galement. La diff€rence entre une structure et une classe est que les attributs et m€thodes d'une structure sont par d€faut publics, alors qu'ils sont par d€faut priv€s dans une classe. Une autre diff€rence est que les structures ne peuvent utiliser l'h€ritage.

Notion de classe

Une classe regroupera donc :

, des donn€es membres.
, des m€thodes membres qui seront des fonctions.

Un premier exemple de classe

, On veut manipuler des points d€finis par une abscisse et une ordonn€e (des r€els).
, Sur un point, on peut calculer la distance entre 2 points et le milieu de 2 points.
, Nous allons donc d€finir une classe Point d€finie 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 d€finit dans ce fichier la classe Point : elle contient 2 donn€es de type double x et y et 2 m€thodes membres distance qui calcule la distance entre ce point et un autre Point et milieu qui calcule le milieu du segment compos€ de ce point et d'un autre Point.

On remarque l'utilisation des directives de compilation #ifndef, #define et #endif pour g€rer 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'impl€mentation de chaque m€thode de la classe Point.
, On fait pr€c€der chaque m€thode de Point::
, On a inclut le fichier cmath afin de pouvoir utiliser la fonction sqrt de cmath (racine carr€e).
, A l'int€rieur de la classe Point, on peut acc€der directement ƒ l'abscisse du point en utilisant la donn€e membre x.
, On peut acc€der ƒ l'abscisse du param„tre P d'une m€thode 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'ordonn€e : "; cin >> A.y;
    cout << endl;
```

```
    cout << "SAISIE DU POINT B" << endl;
    cout << "Tapez l'abscisse : "; cin >> B.x;
    cout << "Tapez l'ordonn€e : "; 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'ordonn€e vaut : " << C.y << endl;
    cout << endl;
    cout << "La distance AB vaut :" << d << endl;
    return 0;
}
```

Explications

, Une fois inclus le fichier d'en-t…te Point.h, on peut d€finir 3 points A, B et C.
, A, B et C sont 3 objets qui sont des instances de la classe Point.
, Les donn€es membres €tant publiques, on peut acc€der *ƒ* l'abscisse et *ƒ* l'ordonn€e de A en dehors de la classe en €crivant A.x et A.y.
, Les m€thodes membres distance et milieu €tant publiques, on peut €crire directement A.milieu(B) ou A.distance(B).

Ex€cution

Lorsqu'on ex€cute ce programme, il s'affiche *ƒ* l'€cran :

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

Encapsulation

, Il faut €viter de donner un acc„s ext€rieur aux donn€es membres d'un objet quelconque.
, On va interdire l'acc„s *ƒ* certaines donn€es membres d'une classe ou certaines m€thodes en utilisant le mot cl€ `private`.
, On ne peut acc€der *ƒ* une variable (ou une m€thode membre) priv€e que par l'int€rieur de la classe.
, par contre, on peut acc€der librement *ƒ* toutes les donn€es membres ou m€thodes membres publiques.
, Cette technique fondamentale permet d'emp…cher au programmeur de faire n'importe quoi : il ne pourra acc€der *ƒ* ces donn€es que par les m€thodes publiques.

Interface et bo•te noire

- Vu de l'ext€rieur, on ne peut acc€der ƒ un objet donn€ que gr†ce ƒ ces m€thodes publiques.
- Ceci permet entre autre de prot€ger l'int€grit€ des donn€es.
- L'ensemble des m€thodes publiques est appel€e l'interface de l'objet.
- De l'ext€rieur, l'objet peut …tre vu comme une bo‡te noire qui poss„de une interface d'acc„s.
- On cache ainsi ƒ l'utilisateur de cette classe comment cette interface est impl€ment€e : seul le comportement de l'interface est important.

Accesseurs et mutateurs

- On pourra acc€der aux valeurs des donn€es membres priv€es gr†ce ƒ des m€thodes sp€cifiques appel€e accesseurs. Les accesseurs seront publics.
- On pourra m…me modifier ces valeurs gr†ce ƒ des fonctions sp€cifiques appel€es mutateurs.
- Cela permet au programmeur d'avoir un contr^le complet sur ces donn€es et sur des contraintes en tout genre qu'il veut imposer ƒ ces donn€es.

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 m€thode void setX(double x) est un mutateur qui permet de modifier la donn€e membre priv€e x.
- Idem pour void setY(double y) avec la donn€e membre priv€e y.
- Les m€thodes double getX() et double getY() sont des accesseurs qui permettent d'acc€der aux valeurs respectives des donn€es membres priv€es x et y.
- Les m€thodes saisir() et afficher() permettent respectivement de saisir et d'afficher les coordonn€es 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'ordonn€e : "; cin >> y;
}

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

Explications

- Dans la m€thode setX(...), il y a une utilisation du mot-cl€ this. Le mot cl€ this d€signe un pointeur vers l'instance courante de la classe elle m…me. this->x d€signe donc la donn€e membre de la classe alors que x d€signe le param„tre de la m€thode 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'acc€der aux donn€es membres x et y sur les instances de Point A, B et C en utilisant A.x ou B.y : il faut obligatoirement passer l*ƒ* une des m€thodes publiques.
- Pour saisir la valeur de A, il suffit d'€crire A.saisir();
- Pour afficher la valeur de A, il suffit d'€crire A.afficher();

Ex€cution

Lorsqu'on ex€cute ce programme, il s'affiche *ƒ* l'€cran :

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

Utiliser les op€rateurs >> et <<

- Pour pouvoir saisir un Point au clavier, on pourrait €crire tout simplement cin >> A; o‰ A est une instance de la classe Point.
- Pour €crire un Point *ƒ* l'€cran, on peut €crire tout simplement cout << B.
- Nous allons utiliser pour cela les fonctions operator>> et operator<<.
- Dans l'exemple 3, on utilisera une mani„re de proc€der assez personnelle sans utiliser de fonctions amies;
- Dans l'exemple 4, nous verrons la m€thode qui semble plus classique bas€e sur les fonctions amies.

Exemple 3 : l'op€rateur >> et l'op€rateur <<

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);
```

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

Explications

, On d€finit une m€thode `operator>>(ostream &out);` qui permet d'afficher le point en utilisant le `ostream out` (le plus souvent ce sera `cout`).
, On d€finit une m€thode `operator<<(istream &in);` qui permet de saisir le point au clavier en utilisant le `istream in` (le plus souvent ce sera `cin`).

Le fichier Point.cpp

Les fonctions setX(), setY(), getX(), getY(), distance() et milieu sont identiques *f* celle de l'exemple 2. On d€finit les 2 op€rateurs de la mani„re suivante :

```
void Point::operator>>(ostream &out)
{
    out << "L'abscisse vaut " << x << endl;
    out << "L'ordonn€e vaut " << y << endl;
}

void Point::operator<<(istream &in)
{
    cout << "Tapez l'abscisse : "; in >> x;
    cout << "Tapez l'ordonn€e : "; in >> y;
}
```

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 << cin;
    cout << endl;
    cout << "SAISIE DU POINT B" << endl;
    B << cin;
    cout << endl;
    C = A.milieu(B);
    d = A.distance(B);
    cout << "MILIEU DE AB" << endl;
    C >> cout;
    cout << endl;
    cout << "La distance AB vaut :" << d << endl;
```

```
    return 0;
}
```

Explications :

On peut directement saisir les coordonn€es d'un point par A<<cin ou l'afficher ƒ l'€cran par A>>cout.

Ex€cution :

Lorsqu'on ex€cute ce programme, il s'affiche ƒ l'€cran :

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

La notation A<<cout

- On peut attacher l'op€rateur de << ƒ la classe Point comme dans l'exemple 3.
- Certains pr€f„rent toutefois €crire de mani„re plus usuelle : cout<<A;
- Ils argumentent parfois en disant qu'en plus cela permet d'encha‡ner les affichages en €crivant : `cout<<A<<endl<<B;`
- Il faudrait donc normalement d€finir une m€thode `operator<<(const Point &A)` sur la classe `ostream`.
- Or la classe `iostream` est d€jƒ €crite !

Les fonctions amies

- Pour pouvoir €crire `cout<<A;` il faut €crire une fonction `ostream & operator<<(ostream &, const Point &P);`
- Le plus pratique est que cette fonction ait le droit d'acc€der aux donn€es membres priv€es de la classe Point.
- On va donc la cr€er en tant que fonction amie en utilisant le mot-cl€ friend.
- L'abus de fonctions amies rompt avec le principe d'encapsulation : ƒ utiliser avec pr€caution.

Exemple 4 : les fonctions amies

Le fichier Point.h

```
#ifndef POINT_H
#define POINT_H
class Point
{
friend istream & operator>>(istream &, Point &P);
friend ostream & operator<<(ostream &, const Point &P);
public:
    void setX(double x);
```

```
    void setY(double y);
    double getX();
    double getY();
    double distance(const Point &P);
    Point milieu(const Point &P);
private:
    double x, y;
};
#endif
```

Explications

, Nous avons d€fini deux fonctions amies de la classe Point : ce ne sont pas des m€thodes membres.
, Leurs d€finitions sont les suivantes :

```
friend istream & operator>>(istream &, Point &P);
friend ostream & operator<<(ostream &, const Point &P);
```

Elles renvoient respectivement une r€f€rence vers un istream et un ostream pour pouvoir encha‡ner par exemple : `cout<<A<<B;`

Le fichier Point.cpp

Voici l'impl€mentation de ces fonctions amies :

```
ostream & operator<<(ostream & out, const Point &P)
{
    out << "L'abscisse vaut " << P.x << endl;
    out << "L'ordonn€e vaut " << P.y << endl;
    return out;
}
istream & operator>>(istream & in,Point &P)
{
   cout << "Tapez l'abscisse : "; in >> P.x;
   cout << "Tapez l'ordonn€e : "; in >> P.y;
   return in;
}
```

Explications

, Nos 2 fonctions renvoient respectivement in et out par une instruction `return` afin de pouvoir encha‡ner les op€rations de saisie et d'affichage.
, `cout` ne modifie pas notre `Point` : on passe donc une r€f€rence vers un `Point` constant.
, Pour `cin`, il faut passer en param„tre une r€f€rence vers un `Point`.

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;
    cin >> A;
    cout << endl << "SAISIE DU POINT B" << endl;
    cin >> B;
    cout << endl;
    C = A.milieu(B);
    d = A.distance(B);
    cout << "MILIEU DE AB" << endl << C << endl;
    cout << "La distance AB vaut :" << d << endl;
    return 0;
}
```

Ex€cution

Lorsqu'on ex€cute ce programme, il s'affiche *ƒ* l'€cran :

```
SAISIE DU POINT A
Tapez l'abscisse : 3.2
Tapez l'ordonn€e : 1.4
SAISIE DU POINT B

Tapez l'abscisse : 5
Tapez l'ordonn€e : 6
MILIEU DE AB

L'abscisse vaut : 4.1
L'ordonn€e vaut : 3.7
La distance AB vaut : 4.93964
```

Constructeurs et initialisation des donn€es membres

- Par d€faut, les donn€es membres d'un objet ne sont pas initialis€es. Il existe toutefois un constructeur par d€faut qui se contente de cr€er ces donn€es membres. Il est appel€ d„s qu'une instance de la classe est cr€e. **Les donn€es membres sont cr€€es mais ne sont pas initialis€es** : elles sont *ƒ* une valeur al€atoire correspondant *ƒ* ce qu'il y avait en m€moire *ƒ* ce moment l*ƒ*.
- Il est vivement recommand€ de d€finir un constructeur par d€faut pour une classe donn€e qui initialisera les donn€es membres : ainsi, il n'existera pas d'instance de la classe avec des donn€es membres non initialis€es.
- On peut €galement d€finir d'autres constructeurs qui permettront d'initialiser les donn€es membres d'un objet avec certaines valeurs.
- Le programmeur sera alors certain que ces donn€es sont dans un €tat coh€rent.
- Le constructeur peut €ventuellement allouer dynamiquement de la m€moire pour une classe complexe : il faut d€sallouer cette m€moire d„s que l'objet n'existe plus.

Les destructeurs

, Le destructeur est appel€ automatiquement d„s qu'un objet est d€truit. Il peut avoir (entre autres) pour rˆle de lib€rer par exemple la m€moire allou€e au cours de l'utilisation de la classe.

Syntaxe des constructeurs et des destructeurs

, Le constructeur par d€faut de la classe A sera not€ A() . Il ne peut rien renvoyer par un return.
, On peut cr€er d'autres constructeur qui seront identifi€s par leurs param„tres : par exemple un constructeur de la classe A peut s'€crire A(int x, int y).
, Le destructeur de la classe A sera not€ ~A(). Il ne renvoie rien par un return et ne peut pas avoir de param„tres.
, Il existe un destructeur par d€faut qui se contente de d€truire les donn€es membres de l'objet.

Exemple 5 : constructeurs et destructeurs

Le fichier Point.h

```
#ifndef POINT_H
#define POINT_H
#include <iostream>
using namespace std;

class Point
{
  public:
    Point();
    Point(double x, double y);
    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);

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

Explications

Par rapport ƒ l'exemple pr€c€dent, nous avons rajout€ 2 constructeurs :

- Le constructeur par d€faut Point();
- Un autre constructeur Point(double x, double y);
- Le constructeur par d€faut va initialiser l'abscisse et l'ordonn€e de notre Point ƒ 0.
- Le second constructeur va initialiser l'abscisse et l'ordonn€e de notre Point respectivement ƒ x et ƒ y.

Ficher Point.cpp

Voici l'impl€mentation des 2 constructeurs :

```
Point::Point()
{
    x = 0;
    y = 0;
}

Point::Point(double x, double y)
{
    this->x = x;
    this->y = y;
}
```

Explications

- Le constructeur par d€faut initialise x et y ƒ la valeur 0.
- Pour le deuxi„me constructeur this->x d€signe la donn€es membre x de la classe et x d€signe le param„tre du constructeur.
- Idem pour this->y.

Fichier main.cpp

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

int main()
{
    Point A, B(3.4, 5.6);
    cout << "Coordonnee du point A :" << endl;
    A >> cout;
    cout << endl << endl;
    cout << "Coordonnee du point B :" << endl;
    B >> cout;
    cout << endl << endl;
    return 0;
}
```

Explications

, Lorsqu'on d€clare un objet par Point A; c'est le constructeur par d€faut de la classe Point qui est appel€ : l'abscisse et l'ordonn€e de A sont initialis€es ƒ 0.
, Lorsqu'on d€clare un objet par Point B(3.4,5.6); c'est le deuxi„me constructeur qui est appel€ l'abscisse de B est initialis€e ƒ 3.4 et l'ordonn€e ƒ 5.6.

Ex€cution

Lorsqu'on ex€cute ce programme, il s'affiche ƒ l'€cran :

```
COORDONNEES DU POINT A
L'abscisse vaut : 0
L'ordonn€e vaut : 0
COORDONNEES DU POINT B
L'abscisse vaut : 3.4
L'ordonn€e vaut : 5.6
```

Liste d'initialisation

, Dans un constructeur, on peut intialiser des donn€es dans la liste d'initialisation : il est d'ailleurs pr€f€rable de les initialiser ƒ cet endroit.
, Certaines donn€es ne peuvent …tre initialis€es qu'ƒ cet endroit : les r€f€rences par exemple.
, Lorsqu'on €tudiera l'h€ritage, nous reparlerons de cette liste d'initialisation.

Exemple 6 : la liste d'initialisation

Nous allons r€€crire les constructeurs des fichiers Point.cpp de l'exemple 5.

Fichier Point.cpp

```
Point::Point() : x(0),y(0)
{
}

Point::Point(double x, double y) : x(x),y(y)
{
}
```

Explications

, Lors de l'€criture d'un constructeur, on peut initialiser une donn€e membre ƒ 0 en €crivant x(0). C'est ce qui est fait dans le constructeur par d€faut Point() pour les donn€es membres x et y.
, Pour le constructeur Point(int x, int y), on initialise la donn€e membre x en €crivant x(x) : le premier x d€signe la donn€e membre de la classe x, le deuxi„me x d€signe le param„tre du constructeur.

Les op€rateurs new et delete

, Si vous avez un pointeur p d€clar€ ainsi : `A* p;` Pour que p pointe vers une nouvelle instance de la classe A, vous pouvez utiliser ainsi l'op€rateur new : `p= new A();`
, Lorsque vous avez cr€€ avec new une nouvelle instance d'un objet, vous …tes tenu de d€truire cet objet avant la fin de votre programme gr†ce ƒ l'op€rateur delete : `delete p;`

new et les constructeurs

, Si la classe A poss„de plusieurs constructeurs par exemple `A();` et `A(int,int);` vous pouvez choisir le constructeur de votre choix :

```
A *x, *y;
x = new A();  // constructeur A();
y = new A(4, 8);  // constructeur A(int, int);
```

, Bien €videmment il ne faudra pas oublier de d€truire les instances de la classe A cr€€es en utilisant delete :

```
delete x;
delete y;
```

Exercices

EXERCICE 1

Šcrire une classe Fraction dont le fichier d'en-t…te est le suivant :

```
#ifndef FRACTION_H
#define FRACTION_H

#include<iostream>
using namespace std;

class Fraction
{
friend ostream & operator<<(ostream & out, const Fraction &f);
friend istream & operator>>(istream &in, Fraction &f);

 public:
    Fraction();
    Fraction(int i);
    Fraction(int num,int den);

    Fraction operator+(const Fraction & f);
    Fraction operator-(const Fraction & f);
    Fraction operator*(const Fraction & f);
    Fraction operator/(const Fraction & f);
 private:
    int num,den;
    int pgcd(int x, int y);
    void normalise();
};

#endif
```

Voici le r^le de chaque fonction :

, Fraction(); : le constructeur par d€faut initialise la fraction *ƒ* 0.

, Fraction(int); : initialise la fraction *ƒ* l'entier i.

- Fraction(int num, int den); : initialise le num€rateur et le d€nominateur de la fraction.
- ostream & operator<<(ostream & out, const Fraction &f) : affiche ƒ l'€cran la fraction f.
- istream & operator>>(istream &in, Fraction &f) : saisit au clavier la fraction f.
- Fraction operator+(const Fraction & f); permet de faire la somme de 2 fractions.
- Fraction operator-(const Fraction & f); permet de faire la diff€rence de 2 fractions.
- Fraction operator*(const Fraction & f); permet de faire la multiplications de 2 fractions.
- Fraction operator/(const Fraction & f); permet de faire la division de 2 fractions.
- int pgcd(int x, int y) : calcule le pgcd de 2 entiers.
- void normalise() : normalise la fraction. Le d€nominateur doit …tre positif et la fraction irr€ductible.

Šcrire un programme principal qui saisit au clavier 2 fractions f1 et f2 et qui affiche E=(f1+3/4-f2)/(f1*f2-5/8)+4.

Sources et contributeurs de l€article

Programmation C++ (d€butant)/Les classes ‹*Source*: http://fr.wikibooks.org/w/index.php?oldid=254565 ‹*Contributeurs*: DavidL, Merrheim, Saamreivax, Sub, Tavernier, Trefleur, 37 modifications anonymes

Licence