

Programmation en C++

Devoir Ecrit – Mardi 30 Mars 2010

Durée : 2 heures

Sans documents, sans calculatrice

*Consignes préalables au devoir : **vous n'utiliserez pas de copie** mais les sections encadrées laissées vides pour répondre aux questions. Il est rappelé qu'une fonction s'appelle également une méthode et qu'un champ s'appelle également un attribut en C++. N'oubliez pas d'indiquer lisiblement votre nom, prénom et groupe !*

NOM

SCHWEITZER

Prénom

Guillaume

Groupe

C.

14,85

Exercice 1 : calcul symbolique avec des polynômes : surcharge d'opérateurs

On souhaite pouvoir travailler avec un nouveau type polynôme permettant de faire des calculs exacts et non approchés. Le début de la rédaction du programme est le suivant, il vous est demandé de compléter les sections laissées vides et encadrées sur le sujet.

polynome.h

```
class polynome
{
    private :
        long          _degre;      // le degre du polynome
        unsigned long *_coeffs;    // le tableau des coefficients

    public :
        polynome(long=0);
        polynome(const polynome &);
        ~polynome();
        friend ostream& operator<<(ostream &os, const polynome &);
        friend istream& operator>>(istream &is, polynome &);
        friend polynome operator+(const polynome &, const polynome &);
        double eval(double x);
};
```

polynome.cpp

```
#include "polynome.h"
polynome::polynome(long degre) : _degre(degre)
{ // code à compléter par vos soins
```

0,2

```
new _coeffs[-degre+1];
```

degré 2:
 $x^2 + x + 1$
 3 coeffs.

```
}
```

```
polynome::~polynome()
{ // code à compléter par vos soins
```

0,75

```
delete[] _coeffs;
```

```
}
```

```

polynome::polynome(const polynome &f):_degre(f._degre)
{
    if (_degre >= 0)
    {
        _coeffs = new long[_degre+1];

        for (long i=0; i <= _degre; i++)
        {
            _coeffs[i] = f._coeffs[i];
        }
    }
    else
    {
        _coeffs = NULL;
    }
}

```

Quel est le nom de la fonction (ou méthode) précédente ?

1

Il s'agit du constructeur par copie

Quand est-elle appelée ?

1

Lorsque l'on veut créer un objet identique à un déjà existant.

Indiquez le sens de la syntaxe utilisée par le paramètre : const polynome &f

1

const : indique que f ne va pas être modifié.
 &f : référence vers f (une référence est un alias).

// surcharge de l'opérateur <<, à vous de jouer

0,65

```

friend ostream & operator << (ostream & os, const polynome & p)
{
    os << "polynome de degré " << p._degre << " et de coeffs : "
    for (int cpt = 0; cpt < p._degre; cpt++)
    {
        os << p._coeffs[cpt] << " ";
    }
    return os;
}

```

// surcharge de l'opérateur +, à vous de jouer

Voir
Page 8!

0,5

```
friend polynome operator + (const polynome & x1, const polynome & x2)
{
    if (x1.degree > x2.degree) {
        for (int cpt=0, cpt < x2.degree, cpt++;) {
            x1.coefs[cpt] += x2.coefs[cpt]; return x1;
        }
    } else {
        for (int cpt=0, cpt < x1.degree, cpt++;) {
            x2.coefs[cpt] += x1.coefs[cpt];
        }
        return x2;
    }
}
```

Ecrivez le prototype de la fonction amie : operator*

friend polynome operator* (const polynome & x1, const polynome & x2);

Ecrivez l'implémentation de la méthode eval

0,5

```
double polynome::eval ( double x)
{
    double result, val = x;
    for (int cpt=0, cpt < degree+1, cpt++;) {
        for (int cpt2=0, cpt2 < degree - cpt, cpt2++;) {
            val = x * val;
        }
        result += coefs[cpt] * val;
        val = 0;
    }
    return result;
}
```

Indiquez si cette méthode est private ou public. Justifiez votre réponse

1

Cette méthode est public car elle se situe après le mot "public:". De plus, ce genre de méthode doit pouvoir être utilisée par l'utilisateur.

Exercice 2 : Templates, conception

On souhaite dans cet exercice créer un conteneur qui est de type "distributeur" : une application obtient des objets à partir d'un guichet unique.

Etape numéro 1 : le distributeur

Soit la classe suivante, utilisée comme distributeur de polynomes

```
class distributeur
{
    private :
        distributeur();
        distributeur(const distributeur &);

        static distributeur *_instance;

        polynome * _avail[10]; 10 polynome par distributeur max.
        unsigned long _nbDistrib; taille utile du tableau.

    public :
        static distributeur *getDistributeur();

        polynome* demanderPoly();
        void rendrePoly();
};
```

Le constructeur de la classe distributeur est `private` : qu'est-ce que cela implique ?

Il ne peut pas être appelé autrement que par une méthode publique. Il est donc impossible de créer un objet en tentant d'accéder à ce constructeur directement.

Les membres `_instance` et `getDistributeur()` sont `static` : que signifie `static` ?

~~static~~ permet de créer de nouvelles instances de la classe sans revenir aux valeurs par défaut. Cela nous sert pour savoir combien d'objets distributeur ont été créés.

Comment peut-on faire un appel à un constructeur `private` ?

On peut faire appel au constructeur `private` grâce à une méthode publique.

voici le code de la méthode `getDistributeur()` :

```

distributeur *distributeur::getDistributeur()
{
    if (_instance == NULL)
    {
        _instance = new distributeur();
    }

    return _instance;
}

```

Quel est son rôle ?

En fonction du contenu de `_instance`, cette méthode va créer un objet de classe `distributeur` en utilisant le constructeur privé.

Combien d'objets `polynome` le distributeur peut-il donner ?

Le distributeur ne peut donner que 10 polynomes.

Ecrivez le code de la fonction `demandePoly()` permettant de récupérer un `polynome` depuis le distributeur. S'il n'y a plus de `polynome` disponible, la fonction retourne `NULL`.

0,5

```

polynome * polynome::demandePoly()
{
    if (nbDistrib == 0) return NULL;
    else
    {
        return -avail[nbDistrib];
        nbDistrib--;
    }
}

```

Ecrivez le code de la redéfinition de la fonction `rendrePoly()` permettant de rendre un `polynome` au distributeur.

0

Etape 2 : indépendance du type stocké

En regardant en détail le code de la classe distributeur, on se rend compte que le type stocké (en l'occurrence la classe polynome) n'a aucune influence sur le comportement du distributeur. Il peut donc être très intéressant de rendre cette classe distributeur indépendantes du type stocké : ceci va être réalisé par le biais de templates.

De plus, on souhaite également pouvoir distribuer un certain nombre d'objets (pas 10 au maximum)

```
template<class T> class distributeur
{
    private :
        distributeur(long size=10);
        distributeur(const distributeur &);

        static distributeur *_instance;

        T **_avail;
        unsigned long _nbAvail; //
        unsigned long _size;

    public :
        static distributeur *getDistributeur();

        T*          demanderObjet();
        void         rendreObjet();
};
```

Quel est le rôle de l'attribut `_avail` ?

↙ Pour créer un tableau dynamique.

Quel est le rôle de l'attribut `_nbAvail` ?

↙ Pour compter la taille utile du tableau

Quel est le rôle de l'attribut `_size` ?

↙ Pour connaître la taille du tableau

Doit-on modifier le code des fonctions `demandeObjet()` et `rendreObjet()` ? (par rapport à `demandePoly()` et `rendrePoly()`). Si oui, indiquez les changements effectués.

Pour ~~demandeObjet~~, non car la taille du tableau n'est pas importante tant qu'on a la valeur taille utile (-nbAvail).

Surcharge de l'opérateur + :

```
friend polynome operator + (const polynome &, const polynome &);
polynome operator + (const polynome & x1, const polynome & x2)
{
    polynome resul;
    if (x1.-degre > x2.-degre) {
        resul.-degre = x1.-degre;
        new resul.-coeffs [x1.-degre + 1];
        dif = x1.-degre - x2.-degre;
        resul.-coeffs = x1.-coeffs;
        for (int cpt = 0, cpt < x2.-degre, cpt++) {
            resul.-coeffs [cpt + dif] = x1.-coeffs [cpt + dif] + x2.-coeffs [cpt];
        }
    }
    else {
        resul.-degre = x2.-degre;
        new resul.-coeffs [x2.-degre + 1];
        dif = x2.-degre - x1.-degre;
        resul.-coeffs = x2.-coeffs;
        for (int cpt = 0, cpt < x1.-degre, cpt++) {
            resul.-coeffs [cpt + dif] = x2.-coeffs [cpt + dif] + x1.-coeffs [cpt];
        }
    }
    return resul;
}
```