université PARIS-SACLAY

Correction TP de programmation n°8

Cours de programmation impérative

—Licence MPI L1 S2 - Info 121—

Exemple d'écriture d'un type abstrait : les polynômes

Dans cette séance de travaux pratiques, nous allons écrire l'implémentation du type abstrait `Polynôme`, défini par les fonctions ci-dessous :

```
void  PolynomeNul(Polynome &p);
bool  egalPoly(Polynome p1, Polynome p2);
void  modifierCoeffPoly(Polynome &p, int d, float co);
int   degrePoly(Polynome p);
float coeffPoly(Polynome p, int d);
bool  estNulPoly(Polynome p);
```

Nous allons remplacer l'implémentation qui avait été fournie la semaine dernière dans les deux fichiers `PolyAbstr.hpp` et `PolyAbstr.cpp` par une nouvelle implémentation.

1 Mise en place

Enregistrez l'archive fournie et décompressez-la. Elle contient les fichiers `Makefile`, `MonPolyAbstr.hpp`, `MonPolyAbstr.cpp`, `ProfPolyAbstr.hpp`, `ProfPolyAbstr.cpp` et `main.cpp`, qui est une correction du TP de la semaine dernière.

Les fichiers `ProfPolyAbstr` contiennent l'implémentation des types abstraits que nous avons utilisée la semaine dernière. Nous allons les remplacer par les fichiers `MonPolyAbstr`, que vous devrez compléter dans la suite de ce TP afin d'obtenir votre propre implémentation du type abstrait `Polynôme`.

Le programme que nous allons exécuter est `main.cpp`. L'architecture du projet est la suivante :

- `main.cpp` inclut `PolyAbstr.hpp` et utilise ses fonctions.
- `PolyAbstr.hpp` contient les déclarations du type concret et des fonctions de manipulation.
- `PolyAbstr.cpp` inclut `PolyAbstr.hpp` et contient les définitions des fonctions de manipulation.

On remarque donc que `main.cpp` inclut `PolyAbstr.hpp` alors qu'il n'y a pas dans l'archive de fichier portant exactement ce nom. Par contre il y a dans l'archive un fichier `ProfPolyAbstr.hpp` et un fichier `MonPolyAbstr.hpp`. Parfois on voudra que `PolyAbstr.hpp` corresponde à `ProfPolyAbstr.hpp` et parfois on voudra qu'il corresponde à `MonPolyAbstr.hpp` (cela vous permettra de comparer les résultats que vous obtenez avec votre implémentation avec ceux qu'on est censé obtenir). Pour indiquer à quel fichier on veut que `PolyAbstr.hpp` corresponde, on utilise des *liens* (qu'on explique dans la suite de l'énoncé).

La première étape est de tester que tout marche bien avec l'implémentation fournie :

1. On veut utiliser l'implémentation de référence `ProfPolyAbstr`. Pour ceci, on va faire les liens suivants :
   — `PolyAbstr.hpp` vers `ProfPolyAbstr.hpp`
   — `PolyAbstr.cpp` vers `ProfPolyAbstr.cpp`
   Ce qui veux dire que si l'on essaye d'accéder au contenu du fichier `PolyAbstr.hpp` on accédera en fait au fichier `ProfPolyAbstr.hpp`. Pour faire ces liens il faut exécuter les commandes :
   ```
   ln -s ProfPolyAbstr.hpp PolyAbstr.hpp
   ln -s ProfPolyAbstr.cpp PolyAbstr.cpp
   ```

La commande «`ln -s`» agit comme une copie.

2. Après avoir exécuté les deux commandes pour faire les liens (données ci-dessus), tapez `make` pour compiler, puis exécutez (`./main`) et vérifiez que tout fonctionne correctement, c'est-à-dire que les affichages obtenus dans le terminal sont cohérents avec le contenu du `main`.
3. On va maintenant basculer sur votre implémentation à vous (`MonPolyAbstr.hpp`). Pour cela, supprimer les anciens liens et en faire de nouveaux avec les commandes suivantes :

```
make clean
rm -f PolyAbstr.hpp PolyAbstr.cpp
ln -s MonPolyAbstr.hpp PolyAbstr.hpp
ln -s MonPolyAbstr.cpp PolyAbstr.cpp
```

4. Faire `make` puis exécuter. Vous devez constater que le résultat n'est plus celui attendu, puisque vous n'avez pas encore écrit votre implémentation (il est même possible que vous ayez besoin de taper `Ctrl C` pour pouvoir interrompre le programme et reprendre la main sur le terminal). Le but de la suite du TP est de faire votre propre implémentation dans les fichiers fournis `MonPolyAbstr`. On retrouvera alors le résultat attendu en exécutant le programme.

2 Calcul avec les polynômes

Attention, dans ce TP les seuls endroits que vous devrez modifier sont les endroits où il y a explicement écrit `/* À écrire */` (un endroit dans `MonPolyAbstr.hpp` et 6 fonctions dans `MonPolyAbstr.cpp`), ainsi que `main.cpp` dans lequel vous devez ajouter des tests de chacune de vos fonctions. Par contre ne supprimez aucune des lignes de code fournies (vous pouvez en commenter certaines temporairement avec `//` ou `/* */`).

▸ **Exercice 1. (Première implémentation)**

Le but est d'implanter les fonctions du type abstrait

```
void  PolynomeNul(Polynome &p);
bool  egalPoly(Polynome p1, Polynome p2);
void  modifierCoeffPoly(Polynome &p, int d, float co);
int   degrePoly(Polynome p);
float coeffPoly(Polynome p, int d);
bool  estNulPoly(Polynome p);
```

avec le type concret suivant :

```
const int MAX_DEGRE = 32;
struct Polynome {
   float coeffs[MAX_DEGRE+1];
};
```

Note : Un polynôme de degré 32 possède 33 coefficients correspondant aux exposants de 0 à 32. D'où le `MAX_DEGRE+1` dans la déclaration.

Note importante : On pourrait être tenté de définir `Polynome` seulement comme un tableau, sans la structure autour, mais cela poserait deux problèmes :

— Pour rester compatible avec le C les tableaux en C++ ont un comportement différent lors d'un passage de paramètre. Ils sont systématiquement passés par référence (pointeur sur le premier élément).

— Le fait d'utiliser une structure facilitera l'amélioration dans la suite.

On vous a déjà fourni les en-têtes et la documentation des fonctions dans le fichier `MonPolyAbstr.hpp`. La définition des fonctions, qui sera dans `MonPolyAbstr.cpp`, va reprendre l'entête mais va préciser cette fois le contenu de la fonction (le corps).

1. Ajouter la définition du type concret dans `MonPolyAbstr.hpp`

```
1  /*
2   * MonPolyAbstr1-corr.hpp
3   */
4
5  #ifndef MONPOLYABSTR1_HPP_INCLUDED
6  #define MONPOLYABSTR1_HPP_INCLUDED
7
8  const int MAX_DEGRE = 20000;
9
10 typedef struct Polynome {
11     float coeffs[MAX_DEGRE+1];
12 } Polynome;
13
14 void  PolynomeNul(Polynome &p);
15 bool egalPoly(Polynome p1, Polynome p2);
16 void  modifierCoeffPoly(Polynome &p, int d, float co);
17 int   degrePoly(Polynome p);
18 float coeffPoly(Polynome p, int d);
19 bool  estNulPoly(Polynome p);
20
21 #endif // MONPOLYABSTR1_HPP_INCLUDED
```

2. Implanter les fonctions du type abtrait dans `MonPolyAbstr.cpp`. Pour tester vos fonctions au fur et à mesure, vous ne pouvez pas utiliser la fonction d'affichage `affichePoly` car elle fait appel non seulement à `coeffPoly`, mais aussi à `degrePoly` et `estNulPoly`. Temporairement avant de pouvoir faire appel à ces fonctions, vous pouvez utiliser la fonction suivante qui n'utilise que `coeffPoly` :

```
void affichePolySimple(Polynome p) {
  for (int i = 0; i <= MAX_DEGRE; i++)
    afficheMonome(i, coeffPoly(p, i), false);
  cout << endl;
}
```

   En plus des tests au fur et à mesure avec des affichages, vous devez aussi tester vos fonctions en utilisant dès que possible les fonctions de test fournies à la fin de `MonPolyAbstr.cpp`, par exemple en ajoutant un appel à ces fonctions de test au début du main de `main.cpp`. Attention au fait que ces fonctions de test utilisent plusieurs fonctions, il faut donc avoir écrit le code de chacune des fonctions appelées par une fonction de test donnée avant de pouvoir l'utiliser (par exemple `test_PolynomeNul` utilise `PolynomeNul`, `estNulPoly` et `coeffPoly`, on ne peut donc pas encore l'utiliser si on a écrit uniquement `PolynomeNul`).

3. Une fois les fonctions implantées et testées, vérifier que tout remarche bien lors de l'exécution du main. Pour cela vous aurez besoin de changer la valeur de `MAX_DEGRE`. En effet le main utilise un polynome de degré 10000 tandis que `MAX_DEGRE` a été fixé dans un premier temps à

32 (le but était que la fonction `affichePolySimple` ne fasse pas trop d'affichages inutiles ; une fois que vous n'avez plus besoin de cette fonction d'affichage, vous pouvez mettre `MAX_DEGRE` à 20000).

✂ ----------

```
1  /*
2   * MonPolyAbstr1-corr.cpp
3   */
4
5  #include <iostream>
6  #include <cmath>
7  #include <cstdlib>
8
9  #include "PolyAbstr.hpp"
10
11 using namespace std;
12
13 void PolynomeNul(Polynome &p) {
14     int i;
15     for (i = 0; i <= MAX_DEGRE; i++)
16         p.coeffs[i] = 0;
17 }
18
19 bool egalPoly(Polynome p1, Polynome p2) {
20   for (int i = 0; i <= MAX_DEGRE; i++)
21     if (p1.coeffs[i] != p2.coeffs[i]) return false;
22   return true;
23 }
24
25 void modifierCoeffPoly(Polynome &p, int d, float co) {
26     if (d < 0 || d > MAX_DEGRE)
27         cerr << __func__ << ": " << d << ": mauvais degre" << endl;
28     else
29         p.coeffs[d] = co;
30 }
31
32 int degrePoly(Polynome p) {
33     int i;
34     for (i = MAX_DEGRE; i >= 0; i--)
35         if (p.coeffs[i]) return i;
36     return -1;
37 }
38
39 float coeffPoly(Polynome p, int d) {
40     if (0 <= d && d <= MAX_DEGRE)
41         return p.coeffs[d];
42     cerr << __func__ << ": " << d << ": mauvais degre" << endl;
43     exit(EXIT_FAILURE);
44 }
45
46 bool estNulPoly(Polynome p) {
47     int i;
48     for (i = 0; i <= MAX_DEGRE; i++)
49         if (p.coeffs[i]) return false;
```

```
50      return true;
51  }
```

▸ **Exercice 2. (Amélioration)**

1. Après avoir validé le bon fonctionnement des fonctions précédentes, une amélioration possible du calcul sur les polynômes est d'intégrer directement au type concret `Polynome`, le degré de ce dernier, comme suit :

   ```
   const int MAX_DEGRE = 32;
   struct Polynome {
      int degre;
      float coeffs[MAX_DEGRE];
   };
   ```

   Redéfinir toutes les fonctions en prenant en compte cette nouvelle structure afin de rendre vos fonctions plus efficaces quand cela est possible.

2. Tester que tout remarche à nouveau.

```
1   /*
2    * MonPolyAbstr2-corr.hpp
3    */
4
5   #ifndef MONPOLYABSTR2_HPP_INCLUDED
6   #define MONPOLYABSTR2_HPP_INCLUDED
7
8   const int MAX_DEGRE = 20000;
9
10  typedef struct Polynome {
11      int degre;
12      float coeffs[MAX_DEGRE+1];
13  } Polynome;
14
15  void  PolynomeNul(Polynome &p);
16  bool egalPoly(Polynome p1, Polynome p2);
17  void  modifierCoeffPoly(Polynome &p, int d, float co);
18  int   degrePoly(Polynome p);
19  float coeffPoly(Polynome p, int d);
20  bool  estNulPoly(Polynome p);
21
22  #endif // MONPOLYABSTR2_HPP_INCLUDED
```

```
/*
 * MonPolyAbstr2-corr.cpp
 */

#include <iostream>
#include <cmath>
#include <cstdlib>

#include "MonPolyAbstr2-corr.hpp"

using namespace std;

void PolynomeNul(Polynome &p) {
    p.degre = -1;
}

void modifierCoeffPoly(Polynome &p, int d, float co) {
    int i;
    if (d < 0 || d > MAX_DEGRE) {
        cerr << __func__ << ": " << d << ": mauvais degre" << endl;
    } else {
        p.coeffs[d] = co;
        if (d > p.degre && co) {
            for (i = p.degre + 1; i < d; i++)
                p.coeffs[i] = 0;
            p.degre = d;
        } else if (d == p.degre && co == 0) {
            for (i = d; i >= 0 && p.coeffs[i] == 0; i--);
            p.degre = i;
        }
    }
}

int degrePoly(Polynome p) {
    return p.degre;
}

float coeffPoly(Polynome p, int d) {
    if (0 <= d && d <= MAX_DEGRE) {
        if (d <= p.degre) return p.coeffs[d];
        return 0;
    }
    cerr << __func__ << ": " << d << ": mauvais degre" << endl;
    exit(EXIT_FAILURE);
}

bool estNulPoly(Polynome p) {
    return p.degre == -1;
}
```