Travaux Pratiques 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 travailler l'implémentation du type abstrait `Polynôme`, définie 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.

L'architecture du projet est la suivante :

- `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.
- `main.cpp` inclut `PolyAbstr.hpp` et utilise ses fonctions.

Les fichiers `ProfPolyAbstr` contiennent l'implémentation des types abstraits que nous avons utilisée la semaine dernière. Nous allons les remplacer par les fichiers `MonPolyAbstr`.

Pour tester que tout marche bien :

1. On veut utiliser l'implémentation de référence. Pour ceci, on va faire les liens suivants :
   — `PolyAbstr.hpp -> ProfPolyAbstr.hpp`
   — `PolyAbstr.cpp -> 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. Faire `make` puis exécuter et vérifier que tout fonctionne correctement.
3. On va maintenant basculer sur votre implémentation à vous. Pour cela, supprimer les anciens liens et en faire de nouveaux avec les commandes suivantes :
   ```
   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. 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

▸ **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'en-tête mais va préciser cette fois le contenu de la fonction (le corps).

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

   Vous pouvez aussi tester vos fonctions en utilisant les fonctions de test fournies à la fin de `MonPolyAbstr.cpp`, par exemple en ajoutant un appel à ces fonctions de test dans le main.

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).

▸ **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.