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 travaillons avec une implantation du type abstrait `Polynôme` vu en cours, définie par les fonctions ci-dessous :

```
void  PolynomeNul(Polynome &p);
void  ModifierCoeffPoly(Polynome &p, int d, float co);
int   DegrePoly(Polynome p);
float CoeffPoly(Polynome p, int d);
bool  EstNulPoly(Polynome p);
float ValeurPoly(Polynome p, float v);
```

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

---

1 Mise en place

Reprendre le projet de la semaine dernière qui doit contenir les trois fichiers : `main.cpp`, `PolyAbstr.hpp` et `PolyAbstr.cpp`. Nous allons remplacer ces deux derniers fichiers par deux fichiers nommés : `MonPolyAbstr.hpp` et `MonPolyAbstr.cpp`. Pour ceci, il faut les créer comme la semaine dernière :

1. Créer un nouveau fichier d'en-tête. Pour ceci :
   — Dans le menu «File» → «New» → «File...», cliquer sur «C++ header».
   — Donner le chemin complet de `MonPolyAbstr.hpp`.
   — Cliquer sur le bouton «All» dans la rubrique «Build Target(s)».
   — Puis cliquer sur «Finish».
2. Créer un nouveau fichier source sous le nom `MonPolyAbstr.cpp` :
   — Dans le menu «File» → «New» → «File...», cliquer sur «C++ source».
   — Donner comme langage «C++».
   — Suivre la même procédure que précédemment avec comme nom `MonPolyAbstr.cpp`.
3. Désactiver les anciens fichiers `PolyAbstr.hpp` et `PolyAbstr.cpp` :
   — Dans la fenêtre «Project», cliquer sur le bouton droit sur le nom du fichier à supprimer et sélectionner «Remove file from project».
   — On pourra ensuite le ré-ajouter en cliquant sur le projet et en sélectionnant «Add file».

Vous pouvez maintenant recopier les en-têtes des fonctions du type abstrait dans `MonPolyAbstr.hpp` et changer les directives `#include "PolyAbstr.hpp"` en `#include "MonPolyAbstr.hpp"`. L'architecture doit être la suivante :

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

2 Calcul avec les polynômes

On reprendra le `main` de la semaine dernière ou celui donné en correction sur le site.

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

1. On demande d'implanter les fonctions du type abstrait

```
void  PolynomeNul(Polynome &p);
void  ModifierCoeffPoly(Polynome &p, int d, float co);
int   DegrePoly(Polynome p);
float CoeffPoly(Polynome p, int d);
bool  EstNulPoly(Polynome p);
float ValeurPoly(Polynome p, float v);
```

   avec le type concret suivant :

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

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.

2. Pour tester vos fonctions au fur et à mesure, vous ne pouvez pas utiliser la fonction d'affichage 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;
}
```

3. Tester que tout remarche bien.

▸ **Exercice 2. (Amélioration)**

4. Si vous avez fini, faire le même travail avec le type concret suivant :

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

5. Tester que tout remarche à nouveau.