Travaux Pratiques de programmation n°6

Cours de programmation impérative

—Licence MPI L1 S2 - Info 121—

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

Dans cette séance de travaux pratiques, nous travaillons avec une implantation du type abstrait `Polynôme` vu en cours, défini 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);
```

L'implantation est fournie dans deux fichiers `PolyAbstr.hpp` et `PolyAbstr.cpp`. Il n'est pas utile que vous compreniez le contenu de ces deux fichiers qui utilisent la programmation orientée objet.

---

1 Conventions de nommage

Quand on commence à écrire des programmes avec quelques dizaines de fonctions, pour éviter d'avoir en permanence à se poser des questions du genre « Quelle est le nom de la fonction qui ... » ou bien « Dans quel ordre dois-je passer les paramètres à la fonction ... », il est utile de suivre quelques conventions. Toutes les entreprises de développement de logiciels, tous les projets de logiciels sérieux fixent ainsi un certain nombre de règles concernant l'indentation du code, le choix des noms des fonctions et l'ordre des paramètres.

Les choix faits dans la convention suivante sont pour la plupart arbitraires, ils sont là seulement pour éviter des hésitations comme par exemple : dois-je écrire «`EstVide`» ou bien «`estVide`» où encore « `est_vide` » ? Rien ne vous interdit de vous en écarter si elles ne vous plaisent pas, mais ne pas suivre de conventions est un bon moyen de perdre bêtement du temps.

- On utilise la convention dite `CamelCase` pour les noms de fonctions (les mots accolés et commençant par une majuscule).
- Les **constructeurs d'un type abstrait** commencent par le nom du type (par exemple `PolynomeNul`) ; le premier paramètre est l'objet à construire.
- Les procédures et fonctions qui manipulent un type abstrait ont leur **nom qui se termine** par le type (ou une abréviation, ici `Poly`).

Quand une procédure fait un calcul, il y a souvent deux manières de transmettre le résultat :

1. soit on **modifie en place** l'un des paramètres passé en mode Donnée-Résultat ; dans ce cas le nom de la fonction sera un **verbe conjugué qui décrit l'action** que l'on fait sur le paramètre ; ce paramètre sera toujours **le premier paramètre**.
   Par exemple `AjoutePoly(p, q)` ajoute `q` au paramètre `p`.
2. soit on **retourne le résultat** dans un paramètre passé en mode Résultat ; dans ce cas le nom de la fonction sera un **nom qui décrit le résultat de l'action**. Le paramètre recevant le résultat sera alors toujours **le dernier paramètre**.
   Par exemple `SommePoly(a, b, c)` place dans `c` la somme `a+b`.

2 Mise en place

Les trois fichiers `main.cpp`, `PolyAbstr.hpp` et `PolyAbstr.cpp` sont à télécharger depuis l'emplacement usuel. Ensuite il faut effectuer les opérations suivantes :

1. Créer un nouveau répertoire «`.../Polynomes`». Ici «`...`» désigne le répertoire où vous avez l'habitude de travailler en TP.
2. Créer un nouveau projet dans le répertoire `.../Polynomes`, et placer le contenu de `main.cpp` dans le projet.
3. Créer un nouveau fichier d'entête. Pour ceci :
   — Dans le menu «File» → «New» → «File...», cliquer sur «C++ header».
   — Donner `.../Polynomes/PolyAbstr.hpp` comme chemin complet.
   — Cliquer sur le bouton «All» dans la rubrique «Build Target(s)».
   — Puis cliquer sur «Finish».
4. Placer le contenu du fichier `PolyAbstr.hpp` de l'archive dans le nouveau fichier créé.
5. Créer un nouveau fichier source sous le nom `PolyAbstr.cpp` :
   — Dans le menu «File» → «New» → «File...», cliquer sur «C++ source».
   — Donner comme langage «C++».
   — Suivre la même procédure que précédemment avec comme nom `PolyAbstr.cpp`.
6. Placer le contenu du fichier `PolyAbstr.cpp` de l'archive dans le nouveau fichier créé.
7. Vous pouvez maintenant sauver tous les fichiers, compiler et exécuter. Vous devriez normalement voir apparaître :

```
Le polynome p est : 4X^5 - 5X^2 + X - 1
Le polynome 3*p est : 12X^5 - 15X^2 + 3X - 3
La derivee de p est : 20X^4 - 10X + 1
```

Ainsi, nous savons créer, afficher, multiplier par une constante, et dériver un polynôme.

On remarque que pour pouvoir les utiliser comme en cours, vous n'avez pas besoin de savoir comment sont implantés les polynômes. Nous utilisons ici des tables associatives qui sont usuellement elles mêmes un type abstrait implanté avec des arbres auto-équilibrants dits rouge-noir.

Attention : Dans la suite de ce TP, on ne modifiera que le fichier `main.cpp`. On considère en effet que le travail d'implantation du type abstrait est effectué par une autre équipe de développement. La semaine prochaine, nous ferons le contraire : on ne modifiera pas le programme principal, mais on implantera nous même les 5 fonctions du type abstrait.

3 Calcul avec les polynômes

En s'inspirant des fonctions déjà écrites on demande d'écrire et de tester dans le `main` les fonctions suivantes (en prenant soin de procéder à chaque étape à des tests de validation) :

1. `float EvalPoly(Polynome p, float x)` qui calcule la valeur du polynôme p au point x. Par exemple, pour le polynôme $X^4 + 2X^2 - 5$, on s'attend à ce que la fonction `EvalPoly` renvoie 19 si elle est évaluée pour $X = 2$. Vous utiliserez pour cela la fonction `Puissance(float x, int degree)`, qui renvoie x^{degree}.
2. `void PolynomeCoeffEgaux(Polynome &p, int degree, float coeff)` est une fonction qui construit un polynôme p de degré `degree` pour lequel tous les coefficients sont égaux à `coeff`. Ainsi, `PolynomeCoeffEgaux(p, 3, 2)` renverra le polynôme $2X^3 + 2X^2 + 2X + 2$.

3. À l'aide de la fonction `PolynomeCoeffEgaux`, construisez un polynôme de degré 10000 dont tous les coefficients sont égaux à 1.0001. Évaluez le polynôme construit à l'aide de la fonction `EvalPoly` au point $X = 1.0001$.

4. L'implémentation de `EvalPoly` est lente. Pour accélérer l'évaluation d'un polynôme, nous allons maintenant implémenter `EvalHornerPoly(Polynome p, float x)`. Cette méthode permet de calculer la valeur d'un polynôme de degré n en ne faisant que n addition et n multiplications au lieu de $\frac{n(n+1)}{2}$. La méthode consiste à multiplier le coefficient de plus haut degré par x, puis à lui ajouter le second coefficient. On multiplie le résultat par x, auquel on ajoute le troisième coefficient, *etc.*

   Considérons l'exemple du polynôme $-2X^3 + 3X^2 - 5X + 6$, pour $X = 3$:
   — Le premier coefficient (celui de plus haut degré), vaut -2.
   — Je le multiple par X et j'ajoute le deuxième coefficient soit 3, et j'obtiens -3.
   — Je multiple à nouveau par X et j'ajoute le troisième coefficient -5, je passe donc à -14.
   — Je multiple une nouvelle fois par X puis j'ajoute le dernier coefficient 6. J'obtiens le résultat final de -36.

5. Si le gain en vitesse n'est pas très élevé, c'est surtout dû à la recherche du coefficient du polynôme qui prend l'essentiel du temps. Cependant, la méthode de Horner possède un autre avantage. Construisez le polynôme $X^{10} - 99X^9$, et évaluez-le pour $X = 100$ avec la méthode normale et la méthode de Horner. Que constatez-vous ?

6. `void ProduitPoly(Polynome p, Polynome q, Polynome &res)` qui place dans `res` le produit de deux polynômes. On utilisera bien évidement les fonctions déjà écrites. Par exemple si l'on fixe P comme suit
$$P = 4X^5 - 5X^2 + X - 1\,,$$
on calcule $P * (X^3 + 2X - 1)$ par
$$\begin{aligned} P * (X^3 + 2X - 1) &= 1 * (P * X^3) + 2 * (P * X^1) + (-1) * P * (X^0) \\ &= (4X^8 - 5X^5 + X^4 - X^3) + 2(4X^6 - 5X^3 + X^2 - X) + (-4X^5 + 5X^2 - X + 1) \\ &= 4X^8 + 8X^6 - 9X^5 + X^4 - 11X^3 + 7X^2 - 3X + 1 \end{aligned}$$

7. `void PuissancePoly(Polynome p, int n, Polynome &res)` qui place dans `res` la puissance n-ième du polynôme `p`. On peut alors afficher les puissances successives du polynôme $(X + 1)$ et voir apparaître le triangle de Pascal :
$$\begin{aligned} &1 \\ &X + 1 \\ &X^2 + 2X + 1 \\ &X^3 + 3X^2 + 3X + 1 \\ &X^4 + 4X^3 + 6X^2 + 4X + 1 \\ &X^5 + 5X^4 + 10X^3 + 10X^2 + 5X + 1 \\ &X^6 + 6X^5 + 15X^4 + 20X^3 + 15X^2 + 6X + 1 \\ &X^7 + 7X^6 + 21X^5 + 35X^4 + 35X^3 + 21X^2 + 7X + 1 \end{aligned}$$