UNIVERSITÉ LIBRE DE BRUXELLES ULB

FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

Langages de programmation 2 : Projet C++

Yves Roggeman *

22 mars 2016

Résumé

Ceci est un énoncé de l'épreuve de seconde session qui se déroule en août ; l'autre concerne le langage Java et est identique à celui de 1re session. Ils servent de base à l'épreuve orale ; l'évaluation finale porte sur l'acquisition des concepts démontrée à cette occasion. Le but de cet exercice de programmation est donc de démontrer une connaissance approfondie et un usage adéquat des constructions du langage de programmation C++, du mécanisme des « templates », de la hiérarchie d'héritages et du polymorphisme. L'évaluation portera donc essentiellement la pertinence des choix effectués sur l'écriture : la codification, la présentation et l'optimisation du programme justifiée par la maîtrise des mécanismes mis en œuvre lors de la compilation et l'exécution du code.

Cet énoncé est une version complétée de celui de janvier ; la partie supplémentaire est mise en évidence.

Le problème posé consiste à écrire une collection de modèles de classes, abstraites ou concrètes, structurée par des relations de dérivation, de composition et d'agrégation permettant *in fine* de définir un type abstrait de « polynôme modulaire », c'est-à-dire de polynômes sur lesquels les opérations arithmétiques sont réalisées modulo un polynôme unitaire [1] fixé. Ceci sera détaillé ci-dessous.

1 La structure des données

1.1 Les vecteurs

On place l'ensemble des types visés comme cas particuliers d'une classe abstraite de vecteur d'éléments d'un type paramétrique. On suppose que les opérateurs arithmétiques usuels sont définis pour ce type des éléments. Pour cette classe abstraite, les opérations d'indice (« `[]` »), les opérateurs « + » et « − », tant unaires que binaires, opérations composante par composante, ainsi que les multiplications par une valeur du type des éléments, tant à gauche qu'à droite, doivent être définies, éventuellement comme méthodes virtuelles pures. On surchargera également les opérateurs d'input et d'output pour cette classe.

On définira également pour les vecteurs l'opération « produit scalaire », surcharge des autres multiplications décrites ci-dessus.

Descendent de cette classe deux versions de modèles de vecteurs concrets : l'une de taille fixée par un paramètre du **template**, l'autre de taille dynamique, information alors contenue dans chaque objet. Les exemples exposés au cours peuvent très largement servir source d'inspiration pour cette partie. Des conversions entre ces deux types doivent être possibles.

On convient que, pour les vecteurs de taille dynamique, les opérations binaires sur deux vecteurs de tailles effectives différentes se limitent logiquement aux composantes présentes dans les deux opérandes, mais que le résultat aura la taille la plus grande, comme si l'opérande de plus petite taille était complété d'éléments nuls aux indices les plus élevés manquants.

1.2 Les polynômes

On conçoit une classe abstraite de polynômes comme cas particulier enrichi de la classe abstraite des vecteurs. En effet, un polynôme est entièrement défini par ses coefficients. Toutefois, la « taille » n'est plus une primitive publique pertinente pour un polynôme, mais bien son *degré*. Ce degré n'est pas égal à la taille du vecteur sous-jacent, mais correspond

*Université libre de Bruxelles (ULB) <yves.roggeman@ulb.ac.be>

1. Un polynôme est dit « *unitaire* » si son coefficient directeur, celui du terme de degré maximal, vaut l'unité.

logiquement à l'indice maximal de composante non nulle. On convient qu'un polynôme nul est de degré négatif ; le degré est donc une valeur signée. Ce degré devra être ajusté lors des opérations effectuées sur les polynômes.

Les opérateurs d'input et d'output seront adaptés pour ce type et on définira également pour les polynômes deux opérations supplémentaires : le produit (opérateur binaire « * ») et l'évaluation (opérateur « `()` » auquel on fournit comme paramètre une valeur du type de base). L'évaluation se fera par un algorithme efficace sans exponentiation (la méthode de Horner par exemple).

Cette classe abstraite de polynôme se particularise en une classe concrète dérivée de la classe de vecteur de taille dynamique. Toute valeur du type des éléments doit être automatiquement convertible en un tel polynôme de degré 0 ou de degré négatif s'il s'agit de la valeur nulle. La construction « triviale » d'un polynôme produit également un tel polynôme nul.

1.3 Les polynômes modulaires

Le but du problème est de définir enfin un modèle de classe de polynômes pour lesquels les opérations sont effectuées modulo un polynôme fixé que nous nommerons ici « diviseur ». On impose que ce polynôme diviseur soit unitaire et de degré au moins 1 : son coefficient directeur, celui du terme de plus haut degré vaut 1 (*i.e.* l'unité, le neutre pour la multiplication des éléments de base).

Le modèle de classe doit être paramétré non seulement par le type de ses coefficients, mais également par le polynôme diviseur. Ce dernier sera donc fourni comme une *référence* à un polynôme *constant* du type de polynôme concret vu ci-dessus. Ce diviseur est bien associé au type défini, non à chaque objet.

Si le polynôme diviseur est unitaire [2] et de degré n, tous les polynômes visés ici seront nécessairement de degré $< n$, puisque ce sont les restes d'une division par ce diviseur. Donc un polynôme modulaire peut être vu comme un cas particulier de vecteur de taille fixe décrit plus haut. Mais ce sont des polynômes, donc ils héritent également des caractéristiques générales d'un polynôme abstrait tel que décrit ci-dessus.

Notons en particulier que l'opérateur binaire « * » signifie bien une multiplication polynomiale, non un produit scalaire…

Pour garantir le respect implicite de cette contrainte, on ne fournira pas au modèle le véritable polynôme diviseur $d(x)$, mais plutôt formellement $d1(x) = x^n - d(x)$ qui est donc un polynôme quelconque de degré strictement inférieur à n. La valeur de n — degré de $d(x)$ qui correspond au nombre de coefficients, donc à la taille du vecteur des coefficients — devra donc aussi être fournie comme paramètre du modèle.

2 Les opérations sur un polynôme modulaire

Nous allons décrire brièvement ici les opérations qui s'appliquent à la classe à construire.

A priori, les opérations particulières sont semblables à celles qui agissent sur des polynômes généraux. En effet, travailler modulo un polynôme diviseur $d(x)$ consiste simplement à remplacer ce résultat habituel $p(x)$ par le polynôme $r(x)$ qui satisfait $p(x) = q(x).d(x) + r(x)$ où $q(x)$ est le quotient de $p(x)$ par $d(x)$ et $r(x)$, le reste. Comme $\deg a(x).b(x) = \deg a(x) + \deg b(x)$ si aucun des deux n'est nul et comme $d(x)$ est unitaire, si $\deg p(x) = d$, on a $\deg q(x) = d - n$ et $\deg r(x) < n$. Dit autrement, cela revient à considérer que « $d(x) \equiv 0$ », ou encore qu'il suffit de remplacer x^n par $d1(x)$, donc plus généralement tout multiple de x^n par le même multiple de $d1(x)$.

D'une manière générale, les opérations qui ne sont donc pas susceptibles d'accroître le degré ne sont pas à spécialiser. C'est le cas de la plupart de celles dont nous avons parlé ici. L'unique exception est la multiplication de deux polynômes modulaires. Mais on peut effectuer la réduction modulaire directement dans l'algorithme modifié, ce qui évite toute création de coefficients supplémentaires (correspondant au degré n ou plus) et toute division explicite de polynômes.

3 Aide

Pour vous aider, voici deux exemples de fonctions réalisant d'une part la multiplication de deux polynômes (vu ici comme simples tableaux d'entiers), d'autre part la multiplication modulaire. Vous pouvez vous en inspirer, mais n'oubliez pas de les adapter à vos propres définitions de types.

2. Pour que cette propriété soit vérifiée, il suffit en fait que le coefficient directeur du diviseur soit inversible dans l'anneau des coefficients.

Programme 1 – Produit de polynômes

```
void mulPol (const int a[], const int da, const int b[], const int db,
             int*& c, int& dc) {
  // Calcule c(x) = a(x).b(x) ; on suppose que les degrés sont exacts
  delete c;
  if (da < 0 || db < 0) {dc = -1; c = new int[1]; c[0] = 0;}
  else { // produit non nul
    dc = da + db; c = new int[dc+1];
    for (int j = 0; j <= db; ++j) c[j] = a[0]*b[j];
    for (int i = 1; i <= da; ++i) {
      for (int j = 0; j < db; ++j) c[i+j] += a[i]*b[j];
      c[i+db] = a[i]*b[db];
    }
  }
}
```

Programme 2 – Produit de polynômes modulaires

```
void mulPolMod (const int a[], const int da, const int b[], const int db,
                const int d1[], const int n, int c[], int& dc) {
  // Calcule c(x) = a(x).b(x) mod xn-d1(x) ; on suppose les données cohérentes
  dc = da < 0 || db < 0 ? -1 : da + db;
  for (int i = 0; i < n && i <= dc; ++i) {              // partie sans modulo
    int ci = 0;
    for (int j = (db<i ? i-db : 0); j <= i && j <= da; ++j) ci += a[j] * b[i-j];
    c[i] = ci;
  }
  for (int i = n; i <= dc; ++i) {                       // i>=n –> i>da, i>db
    int ci = 0, m = 0;
    for (int j = i-db; j <= da; ++j) ci += a[j] * b[i-j];
    for (int j = i-n; j < n; ++j) c[j] += ci*d1[m++];   // n <= i < 2n-1
    for (int j = 0; j < i-n; ++j) c[j] += ci*d1[m++];
  }
  if (dc < n) for (int i = dc+1; i < n; ++i) c[i] = 0; // (utile ?)
  else for (dc = n-1; dc >= 0 && c[dc] == 0; --dc);    // ajuster le degré
}
```

4 Réalisation

Il vous est demandé d'écrire en C++ et de tester les différentes classes décrites. Le programme en instanciera plusieurs versions, tant en dimension, que de type de base — on testera au moins des éléments **int** et **double** — et de diviseur.

D'une manière générale, la concision, la précision, la lisibilité (clarté du texte source), l'efficacité (pas d'opérations inutiles ou inadéquates) et le juste choix des syntaxes typiques de C++ seront des critères essentiels d'appréciation. De brefs commentaires dans le code source sont souhaités pour éclairer les choix de codification. Un schéma « à la UML » de la hiérarchie de vos classes sera également fourni, ainsi qu'un exemple d'output de test.

Votre travail doit être réalisé pour le vendredi 12 août 2016 à 18 heures au plus tard, mais il vous est vivement conseillé de l'achever plus rapidement. Vous remettrez tout votre travail empaqueté en un seul fichier compacté (« .zip » ou autre) via le site du cours (INFO-F-202) sur l'Université virtuelle (http ://uv.ulb.ac.be/). Ceux-ci devront contenir en commentaire vos matricule, nom, prénom et année d'études. Le jour de l'examen, vous viendrez avec une version imprimée — un *listing* — de ces divers fichiers. Une impression du résultat d'une exécution du programme est également demandée.