TD-TP POLYNOME version 2 (13B)

Thème : Enrichissement du paquetage (`P_Poly_V1_G`). Ce travail fait suite à la première étape (simulation d'une calculette polynôme). Il s'agit **d'augmenter** les spécifications de la première version **sans changer** celles qui prévalaient auparavant (**compatibilité ascendante**). Une dernière et troisième séance (TD-TP 17 en semaine 13) de même durée permettra de **tester sérieusement** un paquetage **inconnu** en utilisant les techniques vues au cours sur la **testabilité**.

Concepts (de cette deuxième approche) **: type abstrait de données** T.A.D. (plus complet cette fois), toujours masquage d'information, généricité...etc. Mais, en plus, des notions : de **lecture fichier** (`Get`), **d'accesseurs** (`Get_Degre et Get_Coef`) pour « voir » des composants de la structure, de **modifieur** (`Set_Coef`) pour changer des éléments de la structure de données et enfin une fonction `Valeur` du polynôme pour un `Float` donné. Quelques **exceptions** (mais pas toutes) sont introduites.

Pour améliorer (et/ou compléter) un paquetage donné sans toucher aux premières spécifications on utilise un paquetage fils ici `P_Poly_V1_G.V2` (donc générique aussi). Les fonctionnalités existantes mais complétées avec des exceptions sont redéfinies (surcharge qui posera parfois des problèmes d'ambiguï té ! à voir). Les autres fonctionnalités sont ajoutées.

Méthodes de travail (suivie par les enseignants) **:**

- On **enrichit** les spécifications de la première version paquetage fils (fichier `p_poly_v1_g-v2.ads`). A discuter (cf. page 3). Voir les exceptions, les sous programmes redéfinis et les nouveaux. Peut-on (comme avec `Ecrire`) utiliser **`renames`** pour certains ?
- On **réalise** le corps de `P_Poly_V1_G.V2` avec les exceptions pour commencer dans les 8 sous programmes enrichis. Voyez que `Poly_Reduit` en privé du père est bien commode. Revoir le squelette en TP.
- Pour les exceptions on utilisera `Raise_Exception` bien plus intelligent que **`raise`**. Documentation des exceptions oblige !
- La procédure `Get` mérite d'être étudiée sérieusement. Voir la lecture de plusieurs valeurs (des rationnels) dans un String, la levée des exceptions différenciées. L'utilisation de cette procédure via le clavier passera par Standard_Input.
- On enrichira le programme de test `Ts_Poly2` (où les `Set` sont remplacés par des `Get`) prévoir un bon fichier de données. On testera les exceptions en encapsulant les sous- programmes par un bloc de traitement.

Les exceptions :

Ce sont des exceptions déclarées et non pas des exceptions prédéfinies d'où leur déclaration :

```
Exc_Depassement_Degre,
Exc_Debordement_Coeff,
Exc_Degre_Inexistant,
Exc_Lecture,
Exc_Division_Zero : exception;
```

`Exc_Depassement_Degre` : doit être levée quand un traitement découvre un degré hors des « bornes » admises.
`Exc_Debordement_Coef` : doit être levée lorsqu'un traitement sort des capacités prévues pour le type T_Rationnels.
`Exc_Degre_Inexistant` : doit être levée quand on cherche à accéder à un degré qui n'existe pas dans le polynôme en question.
`Exc_Lecture` : doit être levée pour tout problème lors de la lecture d'un polynôme.
`Exc_Division_Zero` : Doit être levée en cas de division d'un polynôme par un polynôme nul.

Il faut chercher (et discuter) les « endroits » dans le body des fonctions où peuvent avoir lieu ces anomalies. Puis traiter les situations soit en les contrôlant par des **if then ... end if** soit en traitant des exceptions prédéfinies levées naturellement par le système (ce traitement consiste généralement à « personnaliser » l'exception avec un `Raise_Exception`). Les exceptions sur polynômes contraints ne sont pas traitées, pas plus que le contrôle des polynômes utilisés sans être initialisés (on remarque quand même qu'un polynôme déclaré est égal a priori au polynôme nul !).

Rendez (comme pour le premier TP) un listing reformaté du body en indiquant les S/P sérieusement testés.

TP 13B (2 heures)

- Créez un répertoire tp13B. Copiez les fichiers fournis comme d'habitude.
- Réalisez le body du paquetage `P_Poly_V1_G.V2`. Rendez, à la fin (quand vous aurez testé), un listing formaté du body (partez du squelette p_poly_v1_g-v2.adb.squ).
- Voyez et complétez le programme `Ts_Poly2`. Fichier `ts_poly2.adb.squ` (à renommer `ts_poly2.adb`) pour tester les sous-programmes. Comme pour le TP13A il suffit d'instancier les différents paquetages génériques.
- Exécutez le programme `Ts_Poly2`. Indiquez les S/P sérieusement testés. On **redirigera les Sorties écran** dans un fichier `fic_poly.out`. Pas de tests avec des Set tous les polynômes sont dans le fichier `fic_poly.in` (un polynôme par ligne). Testez sérieusement en complétant fic_poly.in. Notez que cela commence par une seule lecture clavier.
- Notion à revoir à la séance n°3 (tests unitaires de la testabilité). Attention **utilisez cette fois des données susceptibles de lever des exceptions**.

```
with Ada.Text_Io; use Ada.Text_Io;

generic
package P_Poly_V1_G.V2 is

   -- on ajoute exceptions et d'autres méthodes (accesseurs et modifieurs).

   -- exceptions :

      Exc_Depassement_Degre,       --  not in 0..Max
      Exc_Debordement_Coef ,         -- not in T_Entier
      Exc_Degre_Inexistant,             -- not in 0..Degre
      Exc_Division_Zero,              -- sans commentaires
      Exc_Lecture
      : exception;
   -- les exceptions sur polynômes contraints ne sont pas traitées
   -- on complète les fonctions (surcharge)
   -- qui vont gérer les exceptions

   function "+" (Poly_A, Poly_B : in T_Polynome) return T_Polynome;
   function "-" (Poly_A, Poly_B : in T_Polynome) return T_Polynome;
   function "-" (Poly_A : in T_Polynome) return T_Polynome;
   function "*" (Poly_A, Poly_B : in T_Polynome) return T_Polynome;
   function "/" (Poly_A, Poly_B : in T_Polynome) return T_Polynome;
   function "rem" (Poly_A, Poly_B : in T_Polynome) return T_Polynome;
   function Derivee (Poly_A : in T_Polynome) return T_Polynome;
   procedure Set (Poly : out T_Polynome; Vect : in T_Liste_Coef);

   -- Entrée-Sortie :
   procedure Put (Poly : in T_Polynome) renames Ecrire;
   procedure Get ( Fich : in File_Type; Poly : out T_Polynome);

   -- accesseurs :
   function Get_Degre (Poly : in T_Polynome) return Natural;
   -- rend le degré le plus élevé du polynôme après réduction

   function Get_Coef(Poly : in T_Polynome; Degre : in Natural)
      return T_Rationnel;
   -- rend la valeur du coefficient de degré DEGRE

   -- modifieur :
   procedure Set_Coef (Poly : in out T_Polynome; Degre : in Natural;
      Valeur : in T_Rationnel);
   -- change la valeur du coefficient de degré DEGRE

   -- autre :
   function Valeur (Poly : in T_Polynome; X : in Float) return Float;
   -- pour un FLOAT X rend la valeur POLY(X) (utile plus tard!)

end P_Poly_V1_G.V2;
```