TD n° 11 P_Rationnels_G

Thème : Ce TD porte, encore, sur le concept de **type abstrait** de données (**T.A.D**), développé en cours (Cours n° 10) et concept déjà mis en œuvre au TD-TP n°10 précédent (cf. P_Pile).

Objectif : Ce TD permet, à nouveau, de montrer comment créer et tester un paquetage ou **module**, regroupant dans une même unité des déclarations : de constantes, de types, d'objets (publics et privés) et de sous-programmes. Il permet de revoir la notion d'**encapsulation** et la notion d'**abstraction** (modèle informatique associé à un élément du monde réel à informatiser).

Travail à réaliser : Il s'agit de construire notamment le paquetage `P_Rationnel_G` que l'on structurera en deux parties (évidemment !) : spécifications et corps.

Dans la partie **spécification** il faut, d'une part, cerner la structure de données idéale pour réaliser (on dit **implémenter**) un rationnel et d'autre part définir les différentes méthodes nécessaires à la manipulation des objets de type `T_Rationnel`. Pour cette première partie, vous vous inspirerez de la réalisation du paquetage P_Pile_4. Le paquetage devra exporter un type `T_Rationnel` qui sera évidemment déclaré **`private`**. D'autre part le numérateur et le dénominateur des rationnels seront « issus » d'un type `T_Entier`. Pour laisser (aux futurs utilisateurs du paquetage) le choix de la déclaration du type `T_Entier` on rendra le paquetage **générique**. Le paquetage sera appelé `P_Rationnels_G` (G pour générique) et le fichier correspondant à la réalisation de la partie spécification sera nommé `p_rationnels_g.ads`.

Ecrivez en TD le texte de la partie spécification qui définira (en partie privée) le type **`T_Rationnel`** comme un type article (`record`) de deux **champs**. Le premier champ (illustrant le numérateur sera de type `T_Entier`, le second (illustrant le dénominateur) sera de sous type `T_Entier_Pos` (à déclarer) restriction de `T_Entier` aux **valeurs strictement positives**.

Cette partie contiendra également les déclarations des diverses méthodes à commencer par :
un constructeur : `function "/" (X : T_Entier; Y : T_Entier) return T_Rationnel;`

cet « outil » va permettre, à l'utilisateur, de manipuler des littéraux ; en effet l'écriture :
`Le_Rationnel := 4/7 ;` est parfaitement correcte (si les littéraux 4 et 7 sont des T_Entier !)
Ce constructeur aura l'objectif de gérer **automatiquement** la réduction de la fraction (à réaliser bien sûr dans le body). Ainsi 8/14 sera réduit en 4/7 par l'opérateur /. De même ce constructeur gérera l'intégrité des données ainsi –4/(-6) stockera 2/3 (de même 4/(-6) stockera –2/3). Enfin l'opérateur lèvera une exception si le deuxième champ est nul. Cet outil, ne s'appliquant pas qu'à des opérandes sous forme littérale, **sera très utile** pour la réalisation du body ! A soigner !

des sélecteurs (utiles puisque l'utilisateur ne peut accéder à la structure) :

```
function Numer  (R : T_Rationnel) return T_Entier;
function Denom (R : T_Rationnel) return T_Entier_Pos;
```

la réalisation prendra une ligne de code (dur !).

des opérateurs de comparaison (puisque le type article est un type à affectation)

```
function "<" ( R1 : T_Rationnel; R2 : T_Rationnel) return  Boolean;
function "<="(R1 : T_Rationnel;  R2 : T_Rationnel) return  Boolean;
function ">" ( R1 : T_Rationnel; R2 : T_Rationnel) return  Boolean;
function ">=" (R1 : T_Rationnel; R2 : T_Rationnel) return  Boolean;
```

on rappelle (cf. cours) que, seul, **un de ces opérateurs** est à réaliser les autres s'en déduisant.

des opérateurs unaires :

```
function  "+" (R: T_Rationnel) return T_Rationnel;
function  "-"  (R: T_Rationnel) return T_Rationnel;
function "abs" (R: T_Rationnel) return T_Rationnel;
```

des opérateurs binaires :

```
function "+" (R1 : T_Rationnel; R2 : T_Rationnel) return T_Rationnel;
function "-" (R1 : T_Rationnel; R2 : T_Rationnel) return T_Rationnel;
function "*" (R1 : T_Rationnel; R2 : T_Rationnel) return T_Rationnel;
function "/" (R1 : T_Rationnel; R2 : T_Rationnel) return T_Rationnel;
function "*" (R1 : T_Rationnel; R2 : T_Entier) return T_Rationnel;
```

ainsi qu'une constante de valeur nulle (0/1) : `Nulle : constant T_Rationnel ;`

Remarque : Il faudra « réduire » certains résultats rationnels dans certaines opérations binaires (à l'aide du constructeur /) d'où la nécessité de disposer de la fonction `Reduit` mais cette fonction devra rester **inaccessible aux utilisateurs** du paquetage **elle sera donc déclarée dans le corps** et **utilisée seulement** par la fonction /.
La structure interne du T.A.D. `T_Rationnel` n'étant pas accessible (type **`private`**), il faudra aussi fournir aux utilisateurs du paquetage des procédures d'entrée et de sortie (E/S) pour les objets de ce type (procédures `Get` et `Put`). Ces procédures seront déclarées dans un paquetage **« fils »** nommé `P_Rationnels_G.Io` (fichier `p_rationnels_g-io.ads`). Les données devront pouvoir être lues à partir du clavier mais également à partir d'un fichier texte. Les spécifications étant maintenant écrites (voir aussi les corrigés), le corps du paquetage devra être réalisé (fichier `p_rationnels_g.adb`).

Un **programme de test** `Ts_Rationnels` (fichier `ts_rationnels.adb`) sera **complété** pour vérifier : la lecture des rationnels, les diverses opérations sur les rationnels et la sortie des différents résultats. Vous **préparerez sur papier des jeux de tests valables** en essayant de couvrir tout le spectre des cas possibles (à montrer à votre enseignant). Les données, pour ce programme de test, ne seront pas saisies à partir du clavier mais lues directement à partir d'un fichier texte. Ceci permettra de **refaire les mêmes tests** en cas de mises au point répétées ! Le fichier, d'identificateur **rationnels.in** (par exemple)**,** qui sera construit à l'aide de l'éditeur de votre choix, contiendra vos jeux d'essai (un rationnel par ligne), les deux composants seront séparés par un Slash (/)). L'absence d'un / (marquant un entier) forcera la lecture à poser le dénominateur à 1 ! Les résultats seront redirigés dans le fichier **rationnels.out**. Ainsi :

```
Duo :~>./ts_rationnels  > rationnels.out
```

TP n° 11 P_Rationnels_G

Objectif : Réalisation et test du corps du paquetage `P_Rationnels_G`. Le paquetage fils générique lui aussi `P_Rationnels_G.Io` est proposé tout fait ! Le TD de deux heures qui précède ce TP (de deux heures également) est supposé fini et … compris !

Méthodes : Créez, à l'aide d'un éditeur, de toute pièce le fichier `p_rationnels_g.adb` **non fourni**. Si vous possédez un générateur de body (cf. AdaGide) n'hésitez pas car les déclarations des sous programmes à réaliser seront générées automatiquement ! Sinon copiez le fichier `p_rationnels_g.ads` sur `p_rationnels_g.adb` et faites les suppressions et modifications qui s'imposent. Par exemple supprimez toutes les déclarations de la structure de données et des types et sous types, insérez le mot **`body`** à sa place et remplacez les ; terminaux des déclarations par **`is`** et finissez les corps avec **`begin … end ;`**. N'oubliez pas `Reduit`.

Les fichiers `p_rationnels_g.ads`, `p_rationnels_g-io.ads`, `p_rationnels_g-io.adb` sont fournis ; insérez seulement votre entête. Comprenez ces modules !

Détails du TP :

- Créez un répertoire tp11.
- Copiez les trois fichiers : `p_rationnels_g.ads`, `p_rationnels_g-io.ads`, `p_rationnels_g-io.adb` et réalisez **`p_rationnels_g.adb`**. **Compilez ce dernier fichier** (`gnatgcc -c ….`) attention pas de `gnatmake` (stupide avec un paquetage !).
- Créez le fichier de test **`ts_rationnels.adb`**. à partir du fichier `ts_rationnels.adb.squ` **en le complétant**.
- Préparez un exécutable avec la commande `gnatmake ts_rationnels`. Le `gnatmake` n'est pas stupide cette fois !
- Préparez le fichier d'entrée **`rationnels.in`**, simple pour commencer puis compliquez le par la suite.
- Exécutez le programme en redirigeant la sortie (`./ts_rationnels > rationnels.out`).
- Vérifier vos résultats en listant le fichier `rationnels.out`.
- Echangez vos fichiers .in. Comparez vos fichiers .out.
- Rendez les listings (formatés avec entête !) des deux fichiers :
  `p_rationnels_g.adb` et `ts_rationnels.adb`.

Attention : comme pour le TP précédent (P_Pile_4) ces paquetages seront utiles pour la suite !

```
-- fichier p_rationnels_g.ads
-- D. Feneuille 98 (d'après un exo GNAT) et T.Avignon
--
-- ceci est un corrigé il doit avoir été conçu en TD !!!

generic
     type T_Entier is range <>; -- paramètre générique entier
package P_Rationnels_G is

   subtype T_Entier_Pos is T_Entier range 1..T_Entier'Last;

   type T_Rationnel is  private; -- Le T.A.D. en question

   Nulle : constant T_Rationnel; --(voir private)

   -- constructeur

   function "/" (X : T_Entier; Y : T_Entier) return T_Rationnel;

   Zero_Denominateur: exception;

   -- sélecteurs (accès aux composants) :

   function Numer (R : T_Rationnel) return T_Entier;
   function Denom (R : T_Rationnel) return T_Entier_Pos;

   -- opérateurs de comparaison

   function "<" (R1 : T_Rationnel; R2 : T_Rationnel) return Boolean;
   function "<="(R1 : T_Rationnel; R2 : T_Rationnel) return Boolean;
   function ">" (R1 : T_Rationnel; R2 : T_Rationnel) return Boolean;
   function ">="(R1 : T_Rationnel; R2 : T_Rationnel) return Boolean;

   -- opérateurs unaires

   function "+"(R: T_Rationnel)   return T_Rationnel;
   function "-"(R: T_Rationnel)   return T_Rationnel;
   function "abs"(R: T_Rationnel) return T_Rationnel;

   -- opérateurs binaires

   function "+"(R1 : T_Rationnel; R2 : T_Rationnel) return T_Rationnel;
   function "-"(R1 : T_Rationnel; R2 : T_Rationnel) return T_Rationnel;
   function "*"(R1 : T_Rationnel; R2 : T_Rationnel) return T_Rationnel;
   function "/"(R1 : T_Rationnel; R2 : T_Rationnel) return T_Rationnel;
   function "*"(R1 : T_Rationnel; R2 : T_Entier)  return T_Rationnel;

private
   type T_Rationnel is record
      Numerateur  : T_Entier  := 0;
      Denominateur: T_Entier_Pos := 1;
   end record;

   Nulle : constant T_Rationnel := (0,1);
end P_Rationnels_G;
```