PROGRAMMATION IMPERATIVE : COMMENT REDIGER LA PARTIE RAFFINAGE DANS UN RAPPORT DE PROJET

Table des matières

1 Introduction

On cherche ici a additionner 2 polynomes a plusieurs variables. Pour cela on va se servir des arbres n-aire pour décrire les polynomes et les additionner efficacement. Dans un premier temps on va chercher un faire un package gńŕique pour les arbre n-aire. Puis on va se servir de ce package pour ecrire un package pour encoder, et additionner des polynome a plusieurs varaibles.

2 Package p_arbre_n

2.1 Definition des types

Pour ce package, j'ai definit 2 exceptions Arbre_inexistant et Arbre_vide J'ai aussi defini le type arbre_n qui est un pointeur vers un type T_noeud. Le type T_noeud est un type enregistrement avec une donnée de type T qui est un type générique et 4 donnée de type arbre_n qui sont le pere, les frere gauche et droit et le premier fils.

2.2 RESOLUTION DU " An_Vide "

2.2.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Vide
-- Sémantique : Détecter si un arbre n-aire est vide ou non
-- Paramètres : a : arbre_n
-- Type retour : booléen (vaut vrai si a est vide)
-- Pre-condition : Aucune
-- Post-condition : vrai si a est vide
-- Erreur : Aucune
------------------------------------------------------------

fonction An_Vide((*D*) a : arbre_n) retourne boolean est
```

2.2.2 R1

début
    (* Détecter si un arbre n ?aire est vide ou non. *)
    retourner a = null
fin

2.3 RESOLUTION DU " An_Creer_Vide "

2.3.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Creer_Vide
-- Sémantique : Créer un arbre n?aire vide
-- Paramètres : /
-- Type retour : arbre_n
-- Pre-condition : Aucune
-- Post-condition : l'arbre est vide
-- Erreur : Aucune
------------------------------------------------------------

fonction An_Creer_Vide retourne arbre_n est
```

2.3.2 R1

début
| (* Créer un arbre n ?aire vide *)
| retourner null
fin

2.4 RESOLUTION DU " An_Valeur "

2.4.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Valeur
-- Sémantique : Retourner la valeur rangée à la racine d'un arbre n-aire
-- Paramètres : a : arbre_n
-- Type retour : T
-- Pre-condition : l'arbre n'est pas vide
-- Post-condition : la valeur est celle de la racine
-- Erreur : Aucune
------------------------------------------------------------

fonction An_Valeur((*D*) a : arbre_n) retourne T est
```

2.4.2 R1

début
| (* Retourner la valeur rangée à la racine d'un arbre n-aire *)
| retourner a.all.data
fin

2.5 RESOLUTION DU " An_Est_Feuille "

2.5.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Est_Feuille
-- Sémantique : Indiquer si un arbre n-aire est une feuille (pas de fils)
-- Paramètres : a : arbre_n
-- Type retour : booléen (vaut vrai si a n'a pas de fils )
-- Pre-condition : Aucune
-- Post-condition : vrai si l'abre est une feuille
-- Erreur : Aucune
------------------------------------------------------------

fonction An_Est_Feuille((*D*) a : arbre_n) retourne boolean est
```

2.5.2 R1

début
| (* Indiquer si un arbre n-aire est une feuille (pas de fils) *)
| retourner a.all.data
fin

2.6 RESOLUTION DU "An_Creer_Feuille "

2.6.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Creer_Feuille
-- Sémantique : Créer un arbre n?aire avec une valeur mais sans fils, ni frère ,
-- ni père
-- Paramètres : nouveau : T
-- Type retour : arbre_n
-- Pre-condition : Aucune
-- Post-condition : l'arbre est une feuille sans pere, ni frere, ni fils
-- Erreur : Aucune
------------------------------------------------------------

fonction An_Creer_Feuille((*D*) nouveau : T) retourne arbre_n est
```

2.6.2 R1

---

début
| (* Créer un arbre n ?aire avec une valeur mais sans fils, ni frère *)
| retourner nouveau T_noeud'(nouveau, null, null, null, null)
fin

2.7 RESOLUTION DU " An_Pere "

2.7.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Pere
-- Sémantique : Retourner l'arbre n?aire père d'un arbre n?aire
-- Paramètres : a : arbre_n
-- Type retour : arbre_n
-- Pre-condition : Aucune
-- Post-condition : retourne le pere de l'arbre
-- Erreur : Arbre_inexistant : si l'arbre n'a pas de pere
------------------------------------------------------------

fonction An_Pere((*D*) a : arbre_n) retourne arbre_n est
```

2.7.2 R1

---

début
| (* Retourner l'arbre n ?aire père d'un arbre n ?aire *)
| **si** *a.all.pere* = *null* **alors**
| | lever Arbre_inexistant
| **sinon**
| | retourner a.all.pere
| **finsi**
fin

2.8 RESOLUTION DU " An_Frere "

2.8.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Frere
-- Sémantique : Retourner le nieme Frere d'un arbre n?aire
--        le numero 1 est le premier frere droit
--        le numero 0 est l'arbre lui?même.
--        le numero ?1 est le premier frere gauche
-- Paramètres : a : arbre_n,
--              n : entier, le numéro du frère
-- Type retour : arbre_n
-- Pre-condition : a non vide
-- Post-condition : retourne le frere demandé suivant n
-- Erreur : Arbre_inexistant : si le frere demandé n'existe pas
------------------------------------------------------------

fonction An_frere((*D*) a: arbre_n; (*D*) n : integer) retourne arbre_n est
```

J'ai choisi un approche recursive pour cette fonction

2.8.2 R1

début
| (* Retourner l'arbre n ?aire père d'un arbre n ?aire *)
| **si** $n = 0$ **alors**
| | (* Cas terminal*)
| | retourner a
| **sinon**
| | rappeler la fonction sur le frere droit ou gauche
| **finsi**
fin

2.8.3 R2

```
début
    (* Retourner l'arbre n ?aire père d'un arbre n ?aire *)
    si n = 0 alors
        (* Cas terminal* )
        retourner a
    sinon
        (*rappeler la fonction sur le frere droit ou gauche*)
        si n > 0 et a.all.frere_d /= null alors
            (*frere droit*)
            return An_frere(a.all.frere_d, n-1) ;
        sinon
            si a.all.frere_g /= null alors
                return An_frere(a.all.frere_g, n+1)
            sinon
                lever Arbre_inexistant
            finsi
        finsi
    finsi
fin
```

2.9 RESOLUTION DU "An_Fils "

2.9.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Fils
-- Sémantique : Retourner le nieme fils de a
--   le numero 1 est le premier fils
--   Attention : le 2ème fils est le frère droit, pas le petit?fils
-- Paramètres : a : arbre_n ,
--              n : entier , le numéro du fils
-- Type retour : arbre_n
-- Pre-condition : a non vide
-- Post-condition : retourne le nième fils de l'abre
-- Erreur : Arbre_inexistant : si le fils demandé n'existe pas
------------------------------------------------------------

fonction An_Fils((*D*) a : arbre_n;(*D*) n : integer) retourne arbre_n est
```

2.9.2 R1

```
début
    (* Retourner le nieme fils de a *)
    si a.all.fils = null alors
        lever Arbre_inexistant
    sinon
        retourne frere_n(a.all.fils, n-1)
    finsi
fin
```

2.10 RESOLUTION DU "An_afficher "

2.10.1 Enoncé

```
------------------------------------------------------------
-- Procédure An_Afficher
-- Sémantique : Afficher le contenu complet d'un arbre n?aire
-- Paramètres : a : arbre_n
-- Pre-condition : Aucune
-- Post-condition : Aucune
-- Erreur : Aucune
------------------------------------------------------------

procedure An_afficher((*D*) a : arbre_n) est
```

2.10.2 R1

début
| (* Afficher le contenu complet d'un arbre n ?aire *)
| **si** *a /= null* **alors**
| | afficher le noeud
| | afficher les fils
| | afficher les freres
| **sinon**
| | rien
| **finsi**
fin

2.10.3 R2

début
| (* Afficher le contenu complet d'un arbre n ?aire *)
| **si** *a /= null* **alors**
| | (*afficher fils*)
| | **si** *a.all.fils /= null* **alors**
| | | An_afficher(a.all.fils)
| | **sinon**
| | | rien
| | **finsi**
| | **si** *a.all.frere /= null* **alors**
| | | An_afficher(a.all.frere_d)
| | **sinon**
| | | rien
| | **finsi**
| **sinon**
| | rien
| **finsi**
fin

2.11 RESOLUTION DU "An_Nombre_Noeuds "

2.11.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Nombre_Noeuds
```

```
-- Sémantique : Retourner le nombre total de noeuds d'un arbre n?aire
-- Paramètres : a : arbre_n
-- Type retour : Entier
-- Pre-condition : Aucune
-- Post-condition : le retour est le nombre de noeud de l'arbre
-- Erreur : Aucune
------------------------------------------------------------

fonction An_Nombre_Noeuds((*D*) a : arbre_n) retourne integer est
```

2.11.2 R1

début
| (* Retourner le nombre total de noeuds d'un arbre n ?aire *)
| **si** *a.all.fils* = *null* **alors**
| | lever Arbre_inexistant
| **sinon**
| | compter fils
| | compter freres
| **finsi**
fin

2.11.3 R2

début
| (* Retourner le nombre total de noeuds d'un arbre n ?aire *)
| **si** *a.all.fils* = *null* **alors**
| | retourner 0
| **sinon**
| | (*compter fils*)
| | **si** *a.all.fils* /= *null* **alors**
| | | count = count + An_Nombre_Noeuds(a.all.fils)
| | **sinon**
| | | rien
| | **finsi**
| | **si** *a.all.frere* /= *null* **alors**
| | | count = count + An_Nombre_Noeuds(a.all.frere_d)
| | **sinon**
| | | rien
| | **finsi**
| **finsi**
fin

2.12 RESOLUTION DU "An_Rechercher"

2.12.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Rechercher
-- Sémantique : Rechercher une valeur dans un arbre n?aire et
-- retourner l'arbre n?aire dont la valeur est racine si elle
-- est trouvée , un arbre n?aire vide sinon
-- Paramètres : a : arbre_n ,
--              data : T, la valeur à rechercher
```

```
-- Type retour : arbre_n
-- Pre-condition : Aucune
-- Post-condition : data est la racine du sous arbre retourné,
--                  si la valeur n'est pas dans l'arbre data est vide
-- Erreur : Aucune
------------------------------------------------------------

fonction An_Rechercher((*D*) a : arbre_n;(*D*) data : T) retourne arbre_n est
```

2.12.2 R1

début
| (* Rechercher une valeur dans un arbre n ?aire et retourner l'arbre n ?aire dont la valeur est racine si elle *)
| **si** *a /= null* **alors**
| | **si** *a.all.data = data* **alors**
| | | resultat trouver
| | **sinon**
| | | rechercher le resultat dans les fils et les freres
| | **finsi**
| **sinon**
| | rien
| **finsi**
fin

2.12.3 R2

(* Rechercher une valeur dans un arbre n ?aire et retourner l'arbre n ?aire dont la valeur est racine si elle *)
si *a /= null* **alors**
| **si** *a.all.data = data* **alors**
| | (*resultat trouver*)
| | resultat = a
| **sinon**
| | (*rechercher le resultat dans les fils et les freres*)
| **finsi**
sinon
| rien
finsi

2.13 RESOLUTION DU "An_Nombre_Noeuds_Valeur"

2.13.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Nombre_Noeuds_Valeur
-- Sémantique : Retourner le nombre de noeuds d'un arbre n?aire
-- dont la valeur est égale à une valeur donnée.
-- Paramètres : a : arbre_n,
--              data : T, la valeur à rechercher
-- Type retour : Entier
-- Pre-condition : Aucune
-- Post-condition : le nombre de noeud avec data comme valeur
```

```
-- Erreur : Aucune
------------------------------------------------------------

fonction An_Nombre_Noeuds_Valeur((*D*)a : arbre_n; (*D*) data : T) return integer est
```

2.13.2 R1

début
| (* Retourner le nombre de noeuds d'un arbre n ?aire dont la valeur est égale à une valeur donnée. *)
| **si** *a.all.fils* = *null* **alors**
| | retourner 0
| **sinon**
| | compter ce noeud ou non
| | compter fils contenant data
| | compter freres contenant data
| **finsi**
fin

2.13.3 R2

début
| (* Retourner le nombre de noeuds d'un arbre n ?aire dont la valeur est égale à une valeur donnée. *)
| **si** *a.all.fils* = *null* **alors**
| | retourner 0
| **sinon**
| | **si** *a.all.data* = *data* **alors**
| | | count = 1
| | **sinon**
| | | count = 0
| | **finsi**
| | **si** *a.all.fils* /= *null* **alors**
| | | count = count + An_Nombre_Noeuds_Valeur(a.all.fils)
| | **sinon**
| | | rien
| | **finsi**
| | **si** *a.all.frere* /= *null* **alors**
| | | count = count + An_Nombre_Noeuds_Valeur(a.all.frere_d)
| | **sinon**
| | | rien
| | **finsi**
| **finsi**
fin

2.14 RESOLUTION DU "An_Est_Racine"

2.14.1 Enoncé

```
------------------------------------------------------------
-- Fonction An_Est_Racine
-- Sémantique : Indiquer si un arbre n?aire est sans père
-- Paramètres : a : arbre_n
-- Type retour : booléen (vaut vrai si a n'a pas de père)
```

```
-- Pre-condition : Aucune
-- Post-condition : vrai si l'abre est une racine
-- Erreur : Aucune
------------------------------------------------------------

fonction An_Est_Racine((*D*) )a : arbre_n) retourne boolean est
```

2.14.2 R1

début
  (* Indiquer si un arbre n ?aire est sans père *)
  si $a = null$ **alors**
    retourn True
  sinon
    retourne a.all.pere = null
  finsi
fin

2.15 RESOLUTION DU "An_Changer_Valeur"

2.15.1 Enoncé

```
------------------------------------------------------------
-- Procédure An_Changer_Valeur
-- Sémantique : Changer la valeur du noeud d'un arbre n?aire
-- Paramètres : a : arbre_n
--              data : T, la nouvelle valeur
-- Pre-condition : Aucune
-- Post-condition : la valeur du noeud de l'arbre est data
-- Erreur : Arbre_vide : si l'arbre en paramètre est vide
------------------------------------------------------------

procedure An_Changer_Valeur((*D/R*) a : arbre_n;(*D*) data : T) est
```

2.15.2 R1

début
  (* Changer la valeur du noeud d'un arbre n ?aire *)
  si $a = null$ **alors**
    lever Arbre_vide
  sinon
    a.all.data = data
  finsi
fin

2.16 RESOLUTION DU "An_Inserer_Fils"

2.16.1 Enoncé

```
------------------------------------------------------------
-- Procédure An_Inserer_Fils
-- Sémantique : Insérer un arbre n?aire sans frère en position de premier
-- fils d'un arbre n?aire a. L'ancien fils de a devient alors le premier
```

```
-- frère de l'arbre n?aire insére.
-- Paramètres : a : arbre_n,
--              a_ins : arbre n, l'arbre à insérer
-- Pre-condition : a_ins n'a pas de frere, a et a_ins ne sont pas vide
-- Post-condition : a_ins est le premier fils de a
-- Erreur : Aucune
------------------------------------------------------------

procedure An_Inserer_Fils((*D/R*) a : arbre_n;(*D*) a_ins : arbre_n) est
```

2.16.2 R1

début
| (* Insérer un arbre n ?aire sans frère en position de premier *)
| **si** *a.all.fils* = *null* **alors**
| | inserer premier fils
| **sinon**
| | ajouter fils et deplacer les autres
| **finsi**
fin

2.16.3 R2

début
| (* Insérer un arbre n ?aire sans frère en position de premier *)
| **si** *a.all.fils* = *null* **alors**
| | (*inserer premier fils*)
| | a_ins.all.pere := a
| | a.all.fils := a_ins
| **sinon**
| | (*ajouter fils et deplacer les autres*)
| | a_ins.all.pere := a
| | a.all.fils.all.frere_g := a_ins
| | a_ins.all.frere_d := a.all.fils
| | a.all.fils := a_ins
| **finsi**
fin

2.17 RESOLUTION DU "An_Inserer_Frere"

2.17.1 Enoncé

```
------------------------------------------------------------
-- Procédure An_Inserer_Frere
-- Sémantique : Insérer un arbre n?aire sans frère en position de premier
-- frère d'un arbre n?aire a
-- Paramètres : a : arbre_n,
--              a_ins : arbre n, l'arbre à insérer
-- Pre-condition : Aucune
-- Post-condition : a_ins est le premier frere de a
-- Erreur : Aucune
------------------------------------------------------------
```

```
procedure An_Inserer_Frere((*D/R*) a : arbre_n; (*D*) a_ins : arbre_n) est
```

2.17.2 R1

début
| (* Insérer un arbre n ?aire sans frère en position de premier *)
| **si** *a.all.frere_d = null* **alors**
| | inserer premier frere
| **sinon**
| | ajouter frere et deplacer les autres
| **finsi**
fin

2.17.3 R2

début
| (* Insérer un arbre n ?aire sans frère en position de premier *)
| **si** *a.all.frere_d = null* **alors**
| | (*inserer premier frere*)
| | a_ins.all.pere := a.all.pere
| | a_ins.all.frere_g := a
| | a.all.frere_d := a_ins
| **sinon**
| | (*ajouter fils et deplacer les autres*)
| | a_ins.all.pere := a.all.pere ; a_ins.all.frere_d := a.all.frere_d
| | a.all.frere_d.all.frere_g := a_ins
| | a_ins.all.frere_g := a
| | a.all.frere_d := a_ins
| **finsi**
fin

2.18 RESOLUTION DU ”supprimer_arbre”

2.18.1 Enoncé

```
------------------------------------------------------------
-- Procédure supprimer_arbre
-- Sémantique : Supprime l'arbre passer en parametre
-- Paramètres : a : arbre_n
-- Pre-condition : Aucune
-- Post-condition : l'arbre a été suprimé
-- Erreur : aucune
------------------------------------------------------------

procedure supprimer_arbre(a : in  arbre_n) is
```

2.18.2 R1

```
début
    (* Supprime l'arbre passer en parametre *)
    si An_Est_Racine(a) alors
        rien
    sinon
        enlever lien du pere
        enlever lien du frere gauche
        enlever lien du frere droit
    finsi
fin
```

2.18.3 R2

```
début
    (* Supprime l'arbre passer en parametre *)
    si An_Est_Racine(a) alors
        rien
    sinon
        (*enlever lien du pere*)
        si a = An_Pere(a).all.fils alors
            An_Pere(a).all.fils := a.all.frere_d
        sinon
            rien
        finsi
        enlever lien du frere gauche
        si a.all.frere_g /= null alors
            a.all.frere_g.all.frere_d := a.all.frere_d
        sinon
            rien
        finsi
        enlever lien du frere droit
        si a.all.frere_d /= null alors
            a.all.frere_d.all.frere_g := a.all.frere_g
        sinon
            rien
        finsi
    finsi
fin
```

2.19 RESOLUTION DU "An_Supprimer_frere"

2.19.1 Enoncé

```
------------------------------------------------------------
-- Procédure An_Supprimer_frere
-- Sémantique : Supprime le nieme frere d'un arbre n?aire
-- Paramètres : a : arbre_n,
--              n : entier, le numéro du frère (et tous ses descendants) à supprimer
-- Pre-condition : Aucune
-- Post-condition : le nieme frere est supprimé
-- Erreur : Arbre_inexistant : si il n'y a pas de nieme frere
------------------------------------------------------------

procedure An_Supprimer_frere((*D/R*)a : arbre_n;(*D*) n : integer) est
```

2.19.2 R1

début
| (* Supprime le nieme frere d'un arbre n ?aire *)
| supprimer_arbre(An_frere(a, n))
fin

2.20 RESOLUTION DU "An_Supprimer_frere"

2.20.1 Enoncé

```
------------------------------------------------------------
-- Procédure An_Supprimer_fils
-- Sémantique : Supprime le nieme fils d'un arbre n?aire
-- Paramètres : a : arbre_n,
--              n : entier, le numéro du fils (et tous ses descendants) à supprimer
-- Pre-condition : Aucune
-- Post-condition : le nieme fils est supprimé
-- Erreur : Arbre_inexistant : si il n'y a pas de nieme fils
------------------------------------------------------------

procedure An_Supprimer_fils((*D/R*) a : arbre_n;(*D*) n : integer) est
```

2.20.2 R1

début
| (* Supprime le nieme fils d'un arbre n ?aire *)
| supprimer_arbre(An_Fils(a, n))
fin

2.21 Tests

J'ai ecrit une procedure de test pour essayer un a un les differentes fonction de ce package. Pour cela j'ai defini 2 variables de arbre_n et je me suis servi d'arbre d'entier avec la fonction d'affichage

```
procedure afficher_T(e:in integer) is
begin
  put(e, 0);
end;
```

Pour chaque test on affiche si le resultat est le bon ou non :

```
arb := An_Creer_Vide;
--test An_vide
put_line(Boolean'image(An_Vide(arb)));

arb := An_Creer_Feuille(8);
--test An_Est_Feuille
put_line(Boolean'image(An_Est_Feuille(arb)));
--test2 An_vide
put_line(Boolean'image(not(An_Vide(arb))));
--test An_Nombre_Noeuds
put_line(Boolean'image((1 = An_Nombre_Noeuds(arb))));
```

```
An_Inserer_Fils(arb, An_Creer_Feuille(5));
--test An_Inserer_Fils
put_line(Boolean'image(An_Est_Feuille(An_Creer_Feuille(5))));
--test2 An_Nombre_Noeuds
put_line(Boolean'image((2 = An_Nombre_Noeuds(arb))));
--test An_pere et An fils
put_line(Boolean'image(An_Pere(An_Fils(arb, 1)) = arb));

An_Inserer_Fils(arb, An_Creer_Feuille(4));
--test2 An_fils et test An_frere
put_line(Boolean'image(An_frere(An_Fils(arb, 1), 1) = An_Fils(arb,2)));
An_afficher(arb);
--test An_Nombre_Noeuds_Valeur
put_line(Boolean'image(An_Nombre_Noeuds_Valeur(arb, 8) = 1));
An_Changer_Valeur(arb, 1);
--test An_changer_valeur
An_afficher(arb);
--test2 An_Nombre_Noeuds_Valeur
put_line(Boolean'image(An_Nombre_Noeuds_Valeur(arb, 8) = 0));
--test An_Est_Racine
put_line(Boolean'image(not(An_Est_Racine(An_Fils(arb, 1)))));
--test2 An_Est_Racine
put_line(Boolean'image(An_Est_Racine(arb)));
```

3 Package Polynome

3.1 Definition des types

Pour ce package, J'ai definit le type T qui est le type des données contenus dans l'arbre. C'est un type enregistrement qui contient 6 champs, var qui contient le nom de la variable, is_var qui est vrai si il y a une variable, exp qui contient l'exposant, is_exp qui est vrai si il y a un exposant, coef qui contient le coefficient et is_coef qui est vrai si il y a un coefficient. J'ai aussi definit le type T_poly qui est du type arbre_n avec le type T et la fonction afficher_T qui sera détailler plus loin.

3.2 RESOLUTION DU "afficher_T"

Pour la fonction encoder, j'ai besoin de lire en entier au milieu d'une chaine de carractère, j'ai donc ecrit une fonction pour ca.

3.2.1 Enoncé

```
------------------------------------------------------------
-- Procedure afficher_T
-- Sémantique : affiche un element de type T
-- Paramètres : e : t, element a afficher
-- Pre-condition : Aucune
-- Post-condition :Aucune
-- Erreur : Aucune
------------------------------------------------------------
procedure afficher_T(e : T) est
```

3.2.2 R1

début
| (* affiche un element de type T *)
| **si** *e.is_coef* **alors**
| | ecrire(e.coef)
| **sinon**
| | rien
| **finsi**
| **si** *e.is_var* **alors**
| | ecrire(e.var)
| **sinon**
| | rien
| **finsi**
| **si** *e.is_exp* **alors**
| | ecrire(e.exp)
| **sinon**
| | rien
| **finsi**
fin

3.3 RESOLUTION DU "lire_entier"

Pour la fonction encoder, j'ai besoin de lire en entier au milieu d'une chaine de carractère, j'ai donc ecrit une fonction pour ca.

3.3.1 Enoncé

```
------------------------------------------------------------
-- Fonction lire_entier
-- Sémantique : lis un entier dans un tableau
-- Paramètres : tab : string, tableau de carractere
--              index : integer, indice de depart
--              n : interger, taille du tableau
-- Type retour : integer
-- Pre-condition : Aucune
-- Post-condition : l'entier retourné est celui dans le tableau
-- Erreur : Aucune
------------------------------------------------------------

fonction lire_entier((*D*) tab : string; (*D/R*) index : integer;(*D*) n : integer) reto
```

3.3.2 R1

début
| (* lis un entier dans un tableau *)
| **tant que** *on lit un chiffre*
| **faire**
| | mettre a jour le resultat
| | incrementer l'index
| **fintantque**
fin

3.3.3 R2

```
début
| (* lis un entier dans un tableau *)
| tant que on lit un chiffre
|  faire
| | (*mettre a jour le resultat*)
| | result := result *10 + (CHARACTER'POS(tab(index))-48)
| | (*incrementer l'index*)
| | index := index + 1
| fintantque
fin
```

3.4 RESOLUTION DU "Encoder "

3.4.1 Enoncé

```
------------------------------------------------------------
-- Fonction encoder
-- Sémantique : Encode un polynome a partir d'un tableau de ccarractere
--en un arbre
-- Paramètres : tab : string, tableau de carractere
-- Type retour : T_poly
-- Pre-condition : Le tableau respecte les conditions de l'enoncé
-- Post-condition : l'arbre correspond au polynome entré
-- Erreur : Aucune
------------------------------------------------------------

fonction encoder((*D*) tab : string;(*D*) n : integer) retourne T_poly est
```

3.4.2 R1

```
début
| (* Encode un polynome a partir d'un tableau de ccarractere. *)
| tant que la chaine n'est pas fini
|  faire
| | traiter le monome
| fintantque
fin
```

3.4.3 R2

```
début
| (* Encode un polynome a partir d'un tableau de ccarractere. *)
| tant que index <= n faire
| | (*traiter le monome*)
| | traiter le coefficient
| | tant que index <=n et tab(index) /= '+' et tab(index) /= '-' faire
| | | construire l'arbre correspondant au monome
| | fintantque
| fintantque
fin
```

3.4.4 R3

```
début
    (* Encode un polynome a partir d'un tableau de ccarractere. *)
    tant que index <= n faire
        (*traiter le monome*)
        (*traiter le coefficient*)
        tant que index <=n et tab(index) /= '+' et tab(index) /= '-' faire
            (*construire l'arbre correspondant au monome*) construire les données creer la
             branche contenant les données
        fintantque
    fintantque
fin
```

3.4.5 R4

```
début
    (* Encode un polynome a partir d'un tableau de ccarractere. *)
    tant que index <= n faire
        (*traiter le monome*)
        (*traiter le coefficient*)
        tant que index <=n et tab(index) /= '+' et tab(index) /= '-' faire
            (*construire l'arbre correspondant au monome*) (*construire les données*) si expo
             = -1 alors
                data := (var => tab(index), is_var => true, exp=>0, is_exp => false, is_coef
                 => false, coef => 0)
            sinon
                data := (var => tab(index), is_var => true, is_exp => true, exp => expo,
                 is_coef => false, coef => 0)
            finsi
            (*creer la branche contenant les données*) si An_Vide(resultat) alors
                result := An_Creer_Feuille(data)
                arb := result
            sinon
                result_rechercher := An_Rechercher(arb, data)
                si not An_vide(result_rechercher) alors
                    cas la branche existe deja
                sinon
                    cas de creation de branche
                finsi
            finsi
        fintantque
    fintantque
fin
```

3.4.6 R4

```
début
    (* Encode un polynome a partir d'un tableau de ccarractere. *)
    tant que index <= n faire
        (*traiter le monome*)
        (*traiter le coefficient*)
        tant que index <=n et tab(index) /= '+' et tab(index) /= '-' faire
            (*construire l'arbre correspondant au monome*) (*construire les données*) si expo
             = -1 alors
                data := (var => tab(index), is_var => true, exp=>0, is_exp => false, is_coef
                 => false, coef => 0)
            sinon
                data := (var => tab(index), is_var => true, is_exp => true, exp => expo,
                 is_coef => false, coef => 0)
            finsi
            (*creer la branche contenant les données*) si An_Vide(resultat) alors
                result := An_Creer_Feuille(data)
                arb := result
            sinon
                result_rechercher := An_Rechercher(arb, data)
                si not An_vide(result_rechercher) alors
                    (*cas la branche existe deja*)
                    arb := result_rechercher
                sinon
                    (*cas de creation de branche*)
                    result_rechercher := An_Creer_Feuille(data)
                    An_Inserer_Fils(arb, result_rechercher)
                    arb := result_rechercher
                finsi
            finsi
        fintantque
    fintantque
fin
```

3.5 RESOLUTION DU "afficher"

3.5.1 Enoncé

```
------------------------------------------------------------
-- procedure afficher
-- Sémantique : affiche un polynome
-- Paramètres : a : T_poly
-- Pre-condition : Aucune
-- Post-condition : vrai si a est vide
-- Erreur : Aucune
------------------------------------------------------------

procedure afficher(a : in T_poly) is
```

3.5.2 R1

début
| (* afficher un polynome *)
| An_afficher(a)
fin

3.6 Tests

Pour tester ce package j'utilise 2 polynomes :

$$X + 2XY$$
$$2XY^2Z^3 + 4YZ^2$$

J'encode ces polynomes et je les affiches Je dois obtenir :

$$Noeud(X, Noeud(1Y, Noeud(0, 1), Noeud(1Y, Noeud(1, 2)))$$
$$Noeud(X, Noeud(1Y, Noeud(2Z, Noeud(3, 2)), Noeud(0Y, Noeud(1Z, Noeud(2, 4)))))$$

4 Conclusion

Je n'ai pas eu le temps de finir ce projet, la partie sur le package p_polynome est tres longue. Ainsi il manque la partie pour décoder un polynome et la partie sur l'addition de 2 polynomes.

Il y aussi des améliorations a faire sur la partie pour encoder un polynôme. Il n'est pas necessaire de garder les variable is_... dans le type T car les les coefficients sont seulement dans les feuilles, les variables partout sauf dans les feuilles et les exposants sont partout sauf a la racine. Donc avec les focntion An_Est_racine et An_Est_Feuille on peut retrouver ces booleans.