Enseignant : M. C. PIOMBO

TRAVAUX PRATIQUES
Correction Série 5

Procédures et Fonctions récursives

NB : Vous devez impérativement utiliser les méthodes de programmation structurée pour développer ces programmes (structure de contrôle, procédure, fonction).

Exercice 1 : Le PGCD

```
------------------------------------------------------------------------------
--
--  Projet : Le PGCD
--
--  Nom Module :  PP.ADB
--  Numéro de version Module : V 1.0
--  But du Module : Fonction récursive qui calcule le PGCD de deux entiers
--       positifs sachant que:
--       Pour trouver le PGCD de deux nombres, on divise le plus grand par
--       le plus petit. Ensuite, on divise le plus petit par le reste
--       de la division précédente, puis on répète le provédé jusqu'au
--       moment où le reste est nul. Le dernier diviseur est le PGCD recherché.
--
--
--  Développeur        /              but de la modif               /  Date
--   C. PIOMBO         /    Création du module                      / 9/04/00
--                     /                                            /
--
--
------------------------------------------------------------------------------
with Ada.Text_Io, Ada.Integer_Text_Io; -- E/S types standards
use Ada.Text_Io, Ada.Integer_Text_Io;

procedure Pp is

   --**********************************
   -- Les Exceptions
   --**********************************

   --**********************************
   -- Définitions des Constantes
   --**********************************

   --**********************************
   -- Définitions des Types Utilisateurs
   --**********************************

   --**********************************
   --          Les Packages
   --**********************************

   --**********************************
   --          Les Procédures
   --**********************************
      ---------------------------------------------------------------------------
   --
   --  Saisir : Saisir une valeur naturelle au clavier
   --           la procédure gère les cas d'erreur par exception
   --           donc la valeur retournée sera obligatoirement valide
   --
   ---------------------------------------------------------------------------
   procedure Saisir (
```

```
         Message : String;        -- Chaine à afficher avant la saisie
         Valeur : out natural     -- valeur saisie au clavier
         ) is

   begin

      loop
         begin
            Put(Message);
            Get(Valeur);Skip_Line;
            exit;
         exception
            -- constraint_error levée pour nombres négatifs
            -- data_error pour lettre et nombre supérieur à natural'last
            when data_error | constraint_error => Skip_Line;
               New_Line;Put_Line("Saisir une valeur naturelle
SVP....");New_Line;
         end;
      end loop;

   end Saisir;

--*********************************
   --          Les Fonctions
   --*********************************

   ------------------------------------------------------------------------------
   --
   --  Pgcd : cette fonction calcule le PGCD de M et N
   --
   --
   ------------------------------------------------------------------------------

   function Pgcd ( M, N : in natural) return natural is
   begin
      if N = 0 then
         return M;
      else
         return Pgcd (N, M rem N);
      end if;
   end Pgcd;

   --*********************************
   --      Les Variables Globales
   --*********************************
   M, N : natural;

begin
   -- ///////////////////////    Corps du programme principal
//////////////////////
   put(natural'last);new_line;

   saisir("Donner une valeur pour m : ", m);
   saisir("Donner une valeur pour n : ", n);

   put ("Le PGCD de "); put(m,3); put(" et "); put(n,3);put(" est : ");
   Put(Pgcd(m , n));

   -- aucune exception à gérer car calcul toujours possible avec les valeurs
   -- renvoyées par saisir()
end Pp;
```

Exercice 2 : La factorielle

```
------------------------------------------------------------------------------
--  Projet : La Factorielle
--
--  Nom Module :  PP.ADB
--  Numéro de version Module : V 1.0
--  But du Module : Fonction récursive qui calcule le produit des entiers
--       positifs de 1 à N sachant que:
--       0! = 1                          ** condition d'arrêt
--       si n>0 alors N! = N * (N-1)!    ** définition de la récursivité
--
--
--  Développeur        /              but de la modif               /  Date
--   C. PIOMBO         /    Création du module                      /  9/04/00
--                     /                                            /
------------------------------------------------------------------------------
with Ada.Text_Io, Ada.Integer_Text_Io; -- E/S types standards
use Ada.Text_Io, Ada.Integer_Text_Io;

procedure Pp is

   --*********************************
   -- Les Exceptions
   --*********************************
--*********************************
   -- Définitions des Constantes
   --*********************************
--*********************************
   -- Définitions des Types Utilisateurs
   --*********************************
--*********************************
   --          Les Packages
   --*********************************
--*********************************
   --          Les Procédures
   --*********************************
   ------------------------------------------------------------------------------
   --
   --  Saisir : Saisir une valeur naturelle au clavier
   --           la procédure gère les cas d'erreur par exception
   --           donc la valeur retournée sera obligatoirement valide
   --
   ------------------------------------------------------------------------------
   procedure Saisir (
         Message : String;        -- Chaine à afficher avant la saisie
         Valeur : out natural     -- valeur saisie au clavier
         ) is

   begin

      loop
         begin
            Put(Message);
            Get(Valeur);Skip_Line;
            exit;
         exception
            -- constraint_error levée pour nombres négatifs
            -- data_error pour lettre et nombre supérieur à natural'last
            when data_error | constraint_error => Skip_Line;
               New_Line;Put_Line("Saisir une valeur naturelle
SVP....");New_Line;
         end;
      end loop;
end Saisir;

   --*********************************
   --          Les Fonctions
   --*********************************
```

```
   ------------------------------------------------------------------------------
   --
   --  Factorielle : fonction qui calcule la factorielle du paramètre N
   --
   --
   ------------------------------------------------------------------------------
   function factorielle ( N : in natural) return positive is
   begin
      if N > 0 then
         return n * factorielle (n-1);
      else
         return 1;
      end if;
   end factorielle;


   --**********************************
   --      Les Variables Globales
   --**********************************
 n : natural;


begin
   -- ///////////////////////    Corps du programme principal
//////////////////////
   put(natural'last);new_line;
   saisir("donner une valeur pour n : ", n);-- lire le plus grand naturel
   put(n,2);put("! :");put(factorielle(n));

   exception
     -- si n = natural'last (2147483647) alors stack overflow
     -- (normal car on empile d'abord les valeurs
     -- avant de faire les calculs en dépilant )
     when storage_error => new_line;put_line("la Pile est pleine !!");
     -- si n = 13 alors débordement de capacité pour les entiers
     when constraint_error => new_line;put_line("debordement de capacite par
calcul");

end Pp;
```

Exercice 3 : Recherche dichotomique d'une occurrence d'une valeur

```
------------------------------------------------------------------------------
--  Projet : Recherche dichotomique d'une occurrence d'une valeur
--
--  Nom Module : PP.ADB
--  Numéro de version Module : V 1.0
--  But du Module :
--          Faire un programme qui recherche l'occurrence d'une valeur saisie
--          au clavier dans un tableau ordonné de 1000 entiers.
--          On fera apparaître dans la solution une procédure récursive qui
--          recevra en paramètre le tableau dans lequel doit s'effectuer
--          la recherche. Cette procédure retournera la place occupée par la
--          première occurrence trouvée de la valeur dans le tableau.
--          Si la valeur recherchée n'existe pas une exception sera levée.
--          Afin d'avoir une suite aléatoire de valeurs entières utiliser
--          la fonction random() du package générique
ada.numerics.discret_Random.
--          NB :
--             On rappelle que rechercher un élément x dans un intervalle [a..b]
--             d'un tableau ordonné T revient à :
--             Tester si T[i] = x pour un i appartenant à l'intervalle [a..b],
** condition d'arrêt
--            Ou bien si T[i] /= x,
--               rechercher x dans l'intervalle [i+1..b]
--               ou dans l'intervalle [a..i-1].     ** définition de la
récursivité
--
--  Développeur        /              but de la modif               /  Date
--    C.PIOMBO         /     Céation du module                      /  9/04/00
--                     /                                            /
--
------------------------------------------------------------------------------
with Ada.Text_Io, Ada.Integer_Text_Io; -- E/S types standards
use Ada.Text_Io, Ada.Integer_Text_Io;

with ada.numerics.discrete_random;

procedure Pp is

   --**********************************
   -- Les Exceptions
   --**********************************
   Recherche_Echouee : exception; -- levée par la fonction de recherche si la
valeur n'existe pas

   --**********************************
   -- Définitions des Constantes
   --**********************************

   --**********************************
   -- Définitions des Types Utilisateurs
   --**********************************

      -- type du tableau dans lequel on fait la recherche
      type Tableau_Entier is array (Positive range 1..1000 ) of Integer;

      -- pointeur sur le tableau utilisé pour le passage de paramètre
      type Ptr_Tab is access all Tableau_Entier;

   --**********************************
   --          Les Packages
   --**********************************
   package p_random is new ada.numerics.discrete_random(integer);
   use p_random;
--**********************************
   --          Les Procédures
   --**********************************
------------------------------------------------------------------------------
```

```
   --
   --  init_tab : initialise le contenu du tableau avec des valeurs aléatoires
   --
   ------------------------------------------------------------------------
   procedure Init_Tab (
         Tab : Ptr_Tab     -- pointeur sur le tableau à initialiser
         ) is

      G : Generator; -- type fourni par le package discrete_random

   begin

      Reset(G); -- débuter le générateur aléatoire dans un état différent à
chaque exécution

      for Indice in Tab'range loop
         Tab(Indice) := Random(G);
      end loop;

   end Init_Tab;

      ------------------------------------------------------------------------
   --
   --  put_line : affiche le contenu du tableau
   --
   ------------------------------------------------------------------------
   procedure Put_Line (
         Tab : Ptr_Tab;     -- pointeur sur le tableau à afficher
         Nbr_Val : Natural            -- le nombre de valeur à afficher
         ) is

   begin
      -- testons la limite
      if Nbr_Val <= Tab'Length then

         for Indice in Tab'First..(Tab'First+Nbr_Val-1) loop
            Put(Tab(Indice));Put(" ");
         end loop;

         New_Line;

      else
         raise Constraint_Error; -- débordement de tableau
      end if;
   end Put_Line;

   ------------------------------------------------------------------------
   --
   --  tri_shell : trier par ordre croissant les données entières contenues
   --           dans un tableau reçu en paramètre par pointeur pour éviter
   --           la copie dans la pile.
   --
   --           utilisation de l'algo du tri Shell
   ------------------------------------------------------------------------
   procedure Tri_Shell(
         Tab : Ptr_Tab;     -- pointeur sur le tableau
         Nbr_Element : Natural  -- nombre d'élément du tableau
         ) is
      Fin : Boolean;
      Ecart, Element2, Tampon : Integer;
   begin
      Ecart := Nbr_Element; -- nbr element
      while Ecart > 1 loop
         Ecart := Ecart/2;
         loop
            Fin := True;
            for Element1 in Tab'First..(Nbr_Element-Ecart) loop
               Element2 := Element1 + Ecart;
```

```
               if Tab(Element1) > Tab(Element2) then
                  Tampon := Tab(Element1);
                  Tab(Element1) := Tab(Element2);
                  Tab(Element2) := Tampon;
                  Fin := False;
               end if;
            end loop;
            exit when Fin;
         end loop;
      end loop;
   end Tri_Shell;


   ------------------------------------------------------------------------------
   --
   --  Saisir : Saisir une valeur entiere au clavier
   --           la procédure gère les cas d'erreur par exception
   --           donc la valeur retournée sera obligatoirement valide
   --
   ------------------------------------------------------------------------------
   procedure Saisir (
         Message : String;        -- Chaine à afficher avant la saisie
         Valeur : out integer     -- valeur saisie au clavier
         ) is

   begin

      loop
         begin
            Put(Message);
            Get(Valeur);Skip_Line;
            exit;
         exception
            -- data_error pour lettre et nombre en dehors de integer'range
            when data_error => Skip_Line;
               New_Line;Put_Line("Saisir une valeur entiere SVP....");New_Line;
         end;
      end loop;

   end Saisir;

   ------------------------------------------------------------------------------
   --
   --  Chercher_Dicho : recherche dichotomique d'une occurrence d'une valeur
   --        entiere X dans un tableau sur un intervalle [A..B]
   --        La procédure gère Echec de la recherche par exception
   --
   ------------------------------------------------------------------------------
   procedure Chercher_Dicho(Tab: ptr_tab; -- tableau où s'effectue la recherche
                            X:Integer; -- Valeur à rechercher
                            A,B : Integer; -- intervalle de recherche
                            I : in out positive) is -- valeur centrale dans
l'intervalle
   begin

      if A <= B then   -- l'intervalle de recherche n'est pas vide
         I := (A+B)/2;
         if Tab(I) /= X then
            if X < Tab(I) then
               Chercher_Dicho (Tab, X ,A, I-1, I);
            else
               Chercher_Dicho (Tab, X , I+1, B, I);
            end if;
         end if;
       else
         raise Recherche_Echouee;
      end if;
   end Chercher_Dicho;
```

```
   --**********************************
   --          Les Fonctions
   --**********************************


   --**********************************
   --      Les Variables Globales
   --**********************************
   Valeur_cherchee : integer;
   Place : positive; -- indice de l'occurrence trouvée
Tableau : aliased Tableau_Entier; -- tableau de recherche

begin
   -- ///////////////////////    Corps du programme principal
-- init du tableau avec des valeurs entières aléatoires
      Init_Tab(Tableau'access);

      -- trier les valeurs dans le tableau
      Tri_Shell(Tableau'access, Tableau'Length);

      -- affichage des valeurs dans le tableau
      Put_Line(Tableau'access, 50);

      -- afficher la dernière valeur dans le tableau pour tests de recherche
      put(tableau(tableau'last) );

      -- saisir la valeur à rechercher
      new_line(2);
      Saisir(" entrez la valeur a rechercher : ",Valeur_Cherchee);

      -- Appel à la procedure chercher_dicho()
      Chercher_dicho(Tableau'access, Valeur_Cherchee,
      tableau'first,Tableau'Last, place);

      -- Affichage du résultat si valeur trouvée
      new_line(2);
      Put("La premiere occurrence de ");Put(Valeur_Cherchee,3);
      Put(" se trouve a la place ");Put(place,3);
      Put(" dans le tableau");

   -- ************* Traitement des erreurs ***************
   exception
       when Recherche_Echouee => Put_Line(" La valeur cherchee est introuvable"
         );
       when Constraint_Error => Put_Line(" acces a l'exterieur du tableau"
         );
      -- par exemple si on passe -1 pour dernier_element


end Pp;
```

Structures de données récursives

Exercice 4 : Calcul polynomial

```
-------------------------------------------------------------------------------
--  Projet : Calcul Polynomial
--
--  Nom Module :  P_POLYNOME.ADS
--  Numéro de version Module : V 1.0
--  But du Module : Interface du paquetage de calcul d'un polynome
--
--  Développeur        /              but de la modif                /  Date
--   C.PIOMBO          /   Création du module                        /  9/04/00
--                     /                                             /
-------------------------------------------------------------------------------
package P_Polynome is

   type ptr_Polynome is private; -- privé pour encapsuler la définition du type

   -- les erreurs
   poly_plein_error : exception; -- plus de place dans le tas
   poly_vide_error : exception; -- aucun terme de définie
   -- degres_dupplique_error : exception; -- un seul terme par degrés, erreur si
duppliqué

   -- creation d'un polynome
   function Creer return ptr_Polynome;

   -- ajouter un terme (coeff et degres) au polynome "poly"
   function Ajouter (Coeff : Integer; Degres : Integer; Poly : ptr_Polynome)
return ptr_polynome;

   -- calculer le polynome "poly"
   function Evaluer (Poly : ptr_Polynome; X : Integer) return Float;

   -- Plus Haut Degrés du Polynome "poly"
   function PHDP (Poly : ptr_Polynome) return integer;

   -- afficher le polynome
   procedure put (poly : in ptr_polynome);

   -- liberer la memoire occupee par le polynome
   procedure Detruire (Poly : in out ptr_Polynome);

private

   -- type de la donnee stockee dans une cellule
   type Terme is
      record
         Coeff  : Integer; -- coeff du terme
         Degres : Integer; -- degres du terme
      end record;

   -- types pour une cellule de la liste chainee
   type Cellule;
   type Ptr_Cellule is access Cellule;
   type Cellule is
      record
         -- donnee d'une cellule
         Donnee : Terme;
         -- lien vers la cellule suivante
         Cellule_Suivante : Ptr_Cellule;
      end record;

   -- la liste chainee
   type Polynome is
      record
         -- pointeur de tete
```

```
         Debut : Ptr_Cellule;
         -- pointeur pour parcourir le polynome
         Courant : Ptr_Cellule;
      end record;

   -- obligation d'utiliser un pointeur pour permettre l'ajout d'un terme
   -- au polynome sous forme de fonction (paramètre en mode in obligatoire).
   type ptr_polynome is access polynome;


end P_Polynome;


--------------------------------------------------------------------------------
--  Projet : Calcul Polynomial
--
--  Nom Module :  P_POLYNOME.ADB
--  Numéro de version Module : V 1.0
--  But du Module : Corps du paquetage de calcul d'un polynome
--      Les coefficients (entiers) et degrés (entiers) de chaque termes seront
--      stockés sous forme d'une liste dynamique simplement chainée.
--
--      NB : . un seul terme par degrés (ajout des coeffs de degré égaux)
--           . être capable de connaitre le terme de plus fort degrés sans
--           parcourir toute la liste chainée.
--
--  Développeur        /              but de la modif               /  Date
--   C.PIOMBO          /   Création du module                       /  9/04/00
--                     /                                            /
--------------------------------------------------------------------------------

with ada.text_io,Ada.Integer_Text_Io;
use ada.text_io,Ada.Integer_Text_Io;

with Ada.Unchecked_Deallocation; -- pour la libération des cellules de liste
chainée

package body P_Polynome is

   -- procedures de desallocation de la memoire dynamique
   procedure Free_Cellule is new Ada.Unchecked_Deallocation(Cellule
      , Ptr_Cellule);
   procedure Free_poly is new Ada.Unchecked_Deallocation(polynome
      , Ptr_polynome);

   --**********************************
   --          Les Fonctions
   --**********************************

   ------------------------------------------------------------------------------
   --
   --  PHDP : retourne la valeur du Plus Haut Degrés du Polynome "poly"
   --
   ------------------------------------------------------------------------------
   function PHDP (Poly : ptr_Polynome) return integer is
   begin

     return Poly.debut.donnee.degres;

   end PHDP;
   ------------------------------------------------------------------------------
   --
   --  Poly_Vide : retourne vrai si le paramètre "Poly" comporte aucun terme
   --              faux sinon
   --
   ------------------------------------------------------------------------------
   function Poly_Vide (Poly : ptr_Polynome ) return Boolean is
   begin
```

```
      return Poly.Debut = null;
   end Poly_Vide;

   ------------------------------------------------------------------------
   --
   --  Ajouter : Ajoute un terme au polynome "Poly".
   --            le terme sera constitué d'un coefficient et de son degrés
   --
   --            Lorsque degrés sera égal à 0 alors on considère qu'on définit
   --            le terme constant
   --
   --            si un terme de degrés équivalent existe, alors
   --              . On ajoute les coefficients des termes
   --              . ou bien une autre solution consiste à interdire
   --                ce cas donc : lever l'exception degres_dupplique_error
   ------------------------------------------------------------------------

   function Ajouter
         (Coeff : Integer; -- coefficient d'un terme
         Degres : Integer; -- degres d'un terme
         Poly : ptr_Polynome) -- le polynome (pointeur car il faut modifier le
poly)
         return ptr_polynome is -- polynome avec le nouveau terme
   begin
      if Poly_Vide( Poly) then
         -- premier terme
         Poly.Debut := new Cellule'( ( Coeff, Degres) , null);
      else
         -- insertion d'un terme de plus grand degres
         if Degres >= Poly.Debut.Donnee.Degres then
           -- terme de degrés identiques ?
           if Degres = Poly.Debut.Donnee.Degres then
              -- raise degres_dupplique_error; -- cas interdit
              Poly.Debut.Donnee.coeff := Poly.Debut.Donnee.coeff + coeff; --
ajout des coeffs
           else
              Poly.Debut := new Cellule'( ( Coeff, Degres) , Poly.Debut);
           end if;
         else
            -- terme de degres < au premier terme
            Poly.Courant := Poly.Debut;
            while Poly.Courant.Cellule_Suivante /= null
                  and then Poly.Courant.Cellule_Suivante.Donnee.Degres
                  > Degres loop
               Poly.Courant := Poly.Courant.Cellule_Suivante;
            end loop;

            -- le terme suivant a un degrés inférieur ou égal au terme à ajouter
            if Poly.Courant.Cellule_Suivante /= null
                  and then Poly.Courant.Cellule_Suivante.Donnee.Degres = Degres
then
                 -- raise degres_dupplique_error; -- un seul terme par degrés
                 Poly.courant.Cellule_Suivante.Donnee.coeff :=
                    Poly.courant.Cellule_Suivante.Donnee.coeff + coeff; -- ajout
des coeffs

            else
                 -- nouveau degrés
                 Poly.Courant.Cellule_Suivante := new Cellule'( ( Coeff
                  , Degres) , Poly.Courant.Cellule_Suivante);
            end if;
         end if;
      end if;

      return poly;

   exception
      when Storage_Error => raise Poly_Plein_Error;
```

```
   end Ajouter;


   ------------------------------------------------------------------------------
   --
   --  Creer : retourne l'adresse de la structure dans le tas définissant
   --          un polynome
   --
   ------------------------------------------------------------------------------
   function Creer return ptr_Polynome is
   begin
      return new polynome'(null,null);

   exception
      when storage_error => raise poly_plein_error;
   end Creer;


   ------------------------------------------------------------------------------
   --
   --  Evaluer : retourne la valeur calculée du polynome en remplaçant x par la
   --            valeur passée en paramètre.
   --
   --            la fonction puissance qui doit être utilisée est celle proposée
   --            par ADA pour les réels car celle pour les entiers n'admet pas
   --            de degrés négatifs.
   --
   --            Attention : Pour définir dans le polynome un terme constant
   --            ce dernier devra avoir le degrés = 0.
   --            ceci pause un problème pour le calcul de 0**0 qui retourne 0.
   --            pour simplifier on dira que se cas ne doit pas être possible
   --            et donc il peut servir pour le formatage de notre terme
constant.
   ------------------------------------------------------------------------------

   function Evaluer (Poly : ptr_Polynome;
                     X : Integer)
                     return Float is -- pour les puissances negatives

      Resultat : Float := 0.0; -- calculs intermediaires
   begin

      -- si aucun terme
      if Poly_Vide( Poly) then
         raise Poly_Vide_Error;
      else

         -- premier terme
         Poly.Courant := Poly.Debut;
```

```
         loop

            if Poly.Courant.Donnee.Degres = 0 then
               -- nous sommes dans le cas particulier du terme constant
               -- il ne faut pas faire appel à la fonction ** car elle retourne
0.0
               resultat := resultat + ( float(Poly.Courant.Donnee.Coeff));

            else
               -- cas général : attention on doit utiliser la version pour les
réels
               -- de la fonction puissance, à cause des degrés négatifs.
               -- 0.0 ** (-1) lèvera l'exception constraint_error
                 Resultat := Resultat +
                     ( float(Poly.Courant.Donnee.Coeff) *
float(X)**Poly.Courant.Donnee.Degres);
            end if;

            Poly.Courant := Poly.Courant.Cellule_Suivante; -- terme suivant

            exit when Poly.Courant = null; -- c'était le dernier terme

         end loop;

      end if;

      return Resultat; -- valeur calculée

   end Evaluer;


   --*********************************
   --          Les Procedures
   --*********************************

   ------------------------------------------------------------------------------
   --
   --  Detruire : Libérer l'espace mémoire occupée par tous les termes du
   --             polynome "Poly"
   --
   ------------------------------------------------------------------------------
   procedure Detruire (Poly : in out ptr_Polynome) is
   begin

      -- Si le poly possède aucun terme
      if Poly_Vide( Poly) then
         raise Poly_Vide_Error;
      else
         -- liberation de l'espace occupe par les cellules
         loop
            Poly.Courant := Poly.Debut.Cellule_Suivante; -- reférence à cellule
suivante
            Free_Cellule(Poly.Debut); -- libérer la première cellule
            exit when Poly.Courant = null; -- plus de terme
            Poly.Debut := Poly.Courant; -- nouveau debut de poly
         end loop;

         free_poly(poly); -- liberation de l'espace occupe par poly

      end if;

   end Detruire;

   ------------------------------------------------------------------------------
   --
   --  Put : Affichage de tous les termes du polynome Poly
   --        Lorsque le degrés est égal à zéro alors c'est le terme constant
```

```
   ------------------------------------------------------------------------------
   procedure Put (Poly : in ptr_Polynome) is
   begin

      -- aucun terme
      if Poly_Vide( Poly) then
         raise Poly_Vide_Error;
      else

         -- premier terme
         Poly.Courant := Poly.Debut;

         loop

            if (Poly.Courant.Donnee.Degres /= 0) then
               Put(Poly.Courant.Donnee.Coeff,3);
               Put(" x**");
               Put(Poly.Courant.Donnee.Degres,1);
            else
               -- terme constant
               Put(Poly.Courant.Donnee.Coeff,3);
            end if;


            Poly.Courant := Poly.Courant.Cellule_Suivante; -- terme suivant
            exit when Poly.Courant = null; -- c'était le dernier terme

            Put(" + ");

         end loop;

      end if;

   end Put;


end P_Polynome;


------------------------------------------------------------------------------
--
--  Projet : Calcul Polynomial
--
--  Nom Module :  PP.ADB
--  Numéro de version Module : V1.0
--  But du Module : programme calculant un polynome de la forme :
--         2x**8 + 5x**6 + 12x**5 + x**3 + 5
--         Permet de définir les coefficients et degrés de chaque terme
--
--  Développeur        /              but de la modif               /  Date
--   C.PIOMBO          /   Création du module                       /  9/04/00
--                     /                                            /
--
--
------------------------------------------------------------------------------
with Ada.Text_Io, Ada.integer_text_io, Ada.float_text_io;-- E/S types standards
use Ada.Text_Io, Ada.integer_text_io, Ada.float_text_Io;

with P_Polynome; -- pour accéder aux fonctions et types
use P_Polynome; -- d'un polynome
procedure Pp is


   --*********************************
   --          Les Procédures
   --*********************************
   ------------------------------------------------------------------------------
   --
```

```
   --  Saisir : Saisir une valeur entière au clavier
   --           la procédure gère les cas d'erreur par exception
   --           donc la valeur retournée sera obligatoirement valide
   --
   ------------------------------------------------------------------------------
   procedure Saisir (
         Message : String;        -- Chaine à afficher avant la saisie
         Valeur : out integer     -- valeur saisie au clavier
         ) is
   begin

      loop
         begin
            Put(Message);
            Get(Valeur);Skip_Line;
            exit;
         exception
            when data_error => Skip_Line;
               New_Line;Put_Line("Saisir une valeur entiere SVP....");New_Line;
         end;
      end loop;

   end Saisir;

--*********************************
   --      Les Variables Globales
   --*********************************
   Mon_Poly : Ptr_Polynome; -- le polynome
   x : integer;

begin
   -- ///////////////////////    Corps du programme principal
//////////////////////

   -- création d'un polynome
   Mon_poly := Ajouter(2, 6, Ajouter(-5, 6, Ajouter ( 3, 0,
               Ajouter ( 2, 0, Ajouter ( 4, 3, Ajouter ( 3, 3,
               Ajouter ( 5, 4, creer)))))));  -- une seule instruction

   Put(Mon_Poly);new_line; -- afficher le polynome, un seul terme par degré

   put("Degres le plus eleve : "); put(PHDP(Mon_poly));new_line;

   -- évaluation du polynome
   saisir("Donner une valeur entiere pour x : ", x);
   put(" Resultat obtenu apres calcul : ");Put(Evaluer(Mon_Poly,x),3,3,0);

   Detruire(Mon_Poly); -- libérer la mémoire dans le tas

   exception
    when poly_vide_error => put_line("Aucun terme dans le polynome");
    when poly_plein_error => put_line("plus de place dans le tas");
    when constraint_error => put_line("Calcul impossible en puissance");
    -- when degres_dupplique_error => put_line("un seul terme par degres");

end Pp;
```

Exercice 5 : évaluation d'une expression arithmétique

```
------------------------------------------------------------------------------
--
--  Projet : Evaluation expression arithmétique préfixée
--
--  Nom Module :  ARBRE_BINAIRE.ADS
--  Numéro de version Module : V 1.0
--  But du Module : Interface du paquetage générique de manipulation des arbres
--                  binaires
--
--  Développeur        /             but de la modif               /  Date
--   C.PIOMBO          /   Création du module                      /  10/04/00
```

```
--   C.PIOMBO          /   Ajout Procédure Destruction Arbre         /  19/04/01
--
------------------------------------------------------------------------------
generic

 type les_donnees is private; -- contenu du noeud de l'arbre

package Arbre_Binaire is
   type ptr_arbre is private; -- pointeur sur un arbre. Type privé afin de
cacher le type noeud
   type ptr_ptr_arbre is access all ptr_arbre; -- définit un pointeur généralisé
                                    -- de pointeur de noeud. Il doit être public
                                    -- car il permet de récupérer l'@ des champs
                                    -- sous_arbre_gauche ou sous_arbre_droit

   -- gestion des erreurs
   Arbre_Plein_Error : exception; -- plus de place dans le tas
   Arbre_Vide_Error  : exception; -- lecture d'un noeud vide dans l'arbre

   -- création d'un arbre vide
   function Creer_Arbre return Ptr_Arbre;
   -- Arbre est il vide
   function Arbre_Vide( Arbre : Ptr_Arbre) return Boolean;
   -- retroune vrai si le noeud est une feuille
   function Feuille(Arbre : Ptr_Arbre) return Boolean;
   -- retourne la valeur de la donnée stockée dans le noeud
   function Lire_Donnee( Arbre : Ptr_Arbre) return les_donnees;
   -- retourne la référence du sous arbre gauche de Arbre
   function Gauche(Arbre : Ptr_Arbre) return Ptr_Arbre;
   -- retourne la référence du champs sous_arbre_gauche de Arbre
   function Ref_Gauche(arbre : ptr_arbre) return ptr_ptr_arbre;
   -- ajoute l'arbre B comme fils gauche de l'arbre A
   procedure Gauche (A, B : ptr_arbre);
   -- retourne la référence du sous arbre droit de Arbre
   function Droite(Arbre : Ptr_Arbre) return Ptr_Arbre;
   -- retourne la référence du champs sous_arbre_droit de Arbre
   function Ref_Droite(arbre : ptr_arbre) return ptr_ptr_arbre;
   -- ajoute l'arbre B comme fils droit de l'arbre A
   procedure Droite (A, B : ptr_arbre);
   -- création d'un noeud de l'arbre
   function Construire(Donnee : les_donnees; Arbre_G, Arbre_D
      : Ptr_Arbre) return Ptr_Arbre;
   -- Destruction de l'arbre binaire
   Procedure detruire_arbre (arbre : in out ptr_arbre);

private

   type Type_Noeud;  -- Permet de définir un pointeur de Noeud sans connaître
son contenu
   type Ptr_Arbre is access Type_Noeud; -- définit un pointeur de Noeud


   -- declaration du type compose récursif Type_Noeud
   type Type_Noeud is
      record
         Donnee : les_donnees;
         -- pointeur sur le type Type_Noeud qui provoque l'aspect récursif
         Sous_Arbre_Gauche,
         Sous_Arbre_Droit  : aliased Ptr_Arbre; -- pour permettre de pointer
dessus
      end record;

end Arbre_Binaire;

------------------------------------------------------------------------------
--
--  Projet : Evaluation expression arithmétique préfixée
--
```

```
--  Nom Module :  ARBRE_BINAIRE.ADB
--  Numéro de version Module : V 1.0
--  But du Module : corps du paquetage générique de manipulation des arbres
--                  binaires
--
--  Développeur        /             but de la modif               /  Date
--   C.PIOMBO          /   Création du module                      /  10/04/00
--   C.PIOMBO          /   Ajout Procédure Destruction Arbre       /  19/04/01
--                     /                                           /
-------------------------------------------------------------------------------
with ada.unchecked_deallocation; -- procédure générique libération du tas

package body Arbre_Binaire is


   --******* DEFINITION DES FONCTIONS DE BASE SUR LES ARBRES BINAIRES
   function Creer_Arbre return Ptr_Arbre is
   begin
      -- création d'un arbre vide
      return null;
   end Creer_Arbre;

   function Arbre_Vide( Arbre : Ptr_Arbre) return Boolean is
   begin
      -- fonction qui teste si l'arbre est vide
      return Arbre = null;
   end Arbre_Vide;


   function Lire_donnee( Arbre : Ptr_Arbre) return les_donnees is
   begin
      -- il faudra gérer a l'extérieur de la function le cas arbre Vide
      -- sinon une erreur sera levée
      if Arbre_Vide(Arbre) then
         raise Arbre_Vide_Error;
      else
         -- retourne la donnée du noeud courant de l'arbre
         return Arbre.all.Donnee;
      end if;
   end Lire_donnee;

   function Gauche(Arbre : Ptr_Arbre) return Ptr_Arbre is
   begin
      -- fournit le sous arbre gauche du noeud courant de l'arbre
      return Arbre.all.Sous_Arbre_Gauche;
   end Gauche;

   function Ref_Gauche(Arbre : Ptr_Arbre) return Ptr_Ptr_Arbre is
   begin
      -- fournit la référence du champs sous_arbre_gauche du noeud courant de
l'arbre
      return Arbre.all.Sous_Arbre_Gauche'access;
   end Ref_Gauche;

   -- ajoute l'arbre B comme fils gauche de l'arbre A
   procedure Gauche (A, B : ptr_arbre) is
   begin
    a.all.Sous_Arbre_Gauche := b;
   end gauche;

   -- ajoute l'arbre B comme fils droit de l'arbre A
   procedure droite (A, B : ptr_arbre) is
   begin
    a.all.Sous_Arbre_droit := b;
   end droite;

   function Droite(Arbre : Ptr_Arbre) return Ptr_Arbre is
   begin
```

```
      -- fournit le sous arbre droit du noeud courant de l'arbre
      return Arbre.all.Sous_Arbre_Droit;
   end Droite;

   function Ref_Droite(Arbre : Ptr_Arbre) return Ptr_Ptr_Arbre is
   begin
      -- fournit la référence du champs sous_arbre_droit du noeud courant de
l'arbre
      return Arbre.all.Sous_Arbre_Droit'access;
   end Ref_Droite;

   function Feuille( Arbre : Ptr_Arbre) return Boolean is
   begin
      -- fonction qui teste si le noeud est une feuille
      return arbre_vide(gauche(Arbre)) and then arbre_vide(droite(Arbre));
   end Feuille;

   function Construire(Donnee : les_donnees; Arbre_G, Arbre_D : Ptr_Arbre)
return Ptr_Arbre is
   begin
      -- construction d'un noeud : Donnée + Sous arbre gauche + Sous arbre droit
      return new Type_Noeud'(Donnee, Arbre_G, Arbre_D);

   exception
     when Storage_Error => raise Arbre_Plein_Error; -- plus de place dans le tas
   end Construire;

   -- Destruction de l'arbre binaire
   procedure free is new ada.unchecked_deallocation(Type_Noeud, ptr_arbre);

   Procedure detruire_arbre (arbre : in out ptr_arbre) is
   begin
    if not arbre_vide(arbre) then

     if not feuille(arbre) then -- test optionnel : evite d'essayer de parcourir
       detruire_arbre(arbre.sous_arbre_gauche); --les sous-arbres d'une feuille!
       detruire_arbre(arbre.sous_arbre_droit);
     end if;

     free(arbre);
    end if;
   end detruire_arbre;

end Arbre_Binaire;
-------------------------------------------------------------------------------
--  Projet : Evaluation expression arithmétique préfixée
--
--  Nom Module :  p_expression.ads
--  Numéro de version Module : V 1.0
--  But du Module : fournir les types et sous_programmes
--                  de manipulation d'une expression arithmétique
--                  préfixée.
--  Développeur        /              but de la modif               /  Date
--   C.PIOMBO          /   Création du module                       /  10/05/00
-------------------------------------------------------------------------------
-- pour accéder aux types et sous_programmes des arbres binaires
with arbre_binaire;

package P_Expression is
   -- gestion des erreurs
   Operateur_Inconnu_Error : exception;
   operande_Error : exception; -- nombre trop grand

   type Ptr_Expression is private; -- evite de mettre en public t_expression

   -- creer une expression dans le tas
   procedure creer_expression (Ptr_exp : out ptr_expression);
   -- détruire une expression
```

```
   procedure detruire_expression(Ptr_exp : out ptr_expression);
   -- saisir une expression
   procedure Saisir (S : String; Ptr_Exp : Ptr_Expression);
   -- nbr caractère dans expression
   function Taille (Ptr_Exp : Ptr_Expression) return Natural;
   -- évaluation de l'expression exp
   function Evaluer( Ptr_Exp : Ptr_Expression ) return Integer;

private
      -- le type des données contenues dans l'arbre
      type Les_Donnees
         (Code_Op : Boolean := True) is
      record
           case Code_Op is
                when True => Operateur : character;
                when False => Operande : Integer; -- feuille de l'arbre
           end case;
      end record;
      -- instance du paquetage générique des arbres binaires
      package arbre_donnees is new arbre_binaire(les_donnees);
      use arbre_donnees;

   -- permet lors des appels récursifs de pointer toujours sur le même objet
   -- et donc de ne pas perdre l'index
   type T_Expression is
      record
  -- chaine de caractères contenant l'expression arithmétique saisie au clavier
         Chaine   : String (1 .. 100);
         -- longueur de l'expression arithmétique
         Longueur : Natural;
         -- indice du dernier caractère traité ( chargé dans l'arbre)
         Index    : Natural           := 0;
         -- arbre de stockage de l'expression
         arbre : aliased ptr_arbre := creer_arbre; -- aliased = pour pourvoir
être pointé par un pointeur de pointeur d'arbre
      end record;

   -- pour manipuler l'expression
   type Ptr_Expression is access all T_Expression;
end P_Expression;
-------------------------------------------------------------------------------
--
--  Projet : Evaluation expression arithmétique préfixée
--
--  Nom Module :  p_expression.adb
--  Numéro de version Module : V 1.0
--  But du Module : fournir les types et sous_programmes
--                  de manipulation d'une expression arithmétique
--                  préfixée.
--
--  Développeur        /              but de la modif               /  Date
--   C.PIOMBO          /   Création du module                       /  10/05/00
--   C.PIOMBO          /   Ajout Procédure Destruction Arbre        /  19/04/01
--                     /                                            /
--
-------------------------------------------------------------------------------
with ada.unchecked_deallocation; -- libération du tas pour l'expression

with ada.text_io;
use ada.text_io;

package body p_expression is
  -- ***************************************************************
  -- *************** procédures privées au paquetage ***************
  -- ***************************************************************

 -- car_suivant : recherche du prochain caractere différent de ' ' et '(' et ')'
   function Car_Suivant ( Expression : ptr_expression) return boolean is
```

```
   begin

      loop
         expression.index := expression.index + 1;
         exit when expression.index > Expression.longueur or else
                   ( Expression.chaine(expression.index) /= ' ' and
                     Expression.chaine(expression.index) /= ')' and
                     Expression.chaine(expression.index) /= '(' );
      end loop;

      -- renvoi indice du prochain caractere /= blanc et '(' et ')'
      -- dans le champ index de expression
      return expression.index <= Expression.longueur; -- plus de caractère ?
   end Car_Suivant;


  -- charger une expression dans un arbre binaire
  procedure charger_expression ( Expression : ptr_expression; Monarbre :
Ptr_Ptr_Arbre ) is

   -- nombre : sommes nous en présence d'une nombre
   function nombre ( Expression : ptr_expression) return boolean is
   begin
    -- on ne peut recevoir qu'un nombre ou un opérateur
    if expression.chaine(expression.index) = '-' then
       return expression.chaine(expression.index + 1) in '0'..'9'; -- nombre
négatif ?
    else
       return expression.chaine(expression.index) in '0'..'9';-- nombre positif?
    end if;
   end nombre;
```

```
 -- fabriquer un nombre entier
 function fab_nombre ( Expression : ptr_expression) return integer is
  tampon : string(1..integer'width); -- nombre de digit du plus grand entier
  i : positive range 1..integer'width := 1; -- indice du tampon
 begin

    loop
      tampon(i) := expression.chaine(expression.index);
      -- on prend le caractère qui suit
      expression.index := expression.index + 1;
      exit when expression.chaine(expression.index) not in '0'..'9';
          i := i + 1;
    end loop;

    return integer'value(tampon(1..i)); -- on ne prend que les caractères utiles

    exception
      -- debordement de tableau
      when constraint_error => raise operande_Error; -- nombre trop grand
 end fab_nombre;

-- les variables locales
un_nombre : Les_Donnees(false);

begin
 -- recherche du prochain caractère à traiter
 if car_suivant(expression) then
     -- quel est le type du caractère à traiter
     if nombre(expression) then
     -- ************* c'est un nombre ==> Alors c'est une feuille
     -- recupérer le nombre et le stocker dans l'arbre
        un_nombre.operande := fab_nombre(expression);
        Monarbre.all := construire ( un_nombre , Creer_Arbre, Creer_Arbre );
     else
     -- ************* c'est un opérateur

     -- mettre opérateur dans l'arbre
        Monarbre.all := construire ( (true, expression.chaine(expression.index))
                                   , Creer_Arbre, Creer_Arbre);
     -- charger l'opérande gauche sur cet opérateur
        charger_expression(expression, Ref_gauche(Monarbre.all) ); -- ici on
                                                                   -- utilise les
     -- charger l'opérande droit sur cet opérateur
        charger_expression(expression, Ref_droite(Monarbre.all) ); -- pointeurs
                                                                   -- de pointeur
     end if;
 end if;
end charger_expression;

-- evaluer l'expression en parcourant l'arbre
function evaluer ( monarbre : ptr_arbre ) return integer is
  -- calculer x op.operateur y
  function calcul ( op : les_donnees; x,y : integer) return integer is
   resultat : integer;
  begin
    case op.operateur is
      when '*' => resultat := ( x * y );
      when '+' => resultat := ( x + y );
      when '-' => resultat := ( x - y );
      when '/' => resultat := ( x / y );
      when others => raise operateur_inconnu_Error;
    end case;
    return resultat;
  end calcul;
donnee : les_donnees;
x, y : integer;

begin
```

```
  donnee := Lire_Donnee(monarbre);

  if feuille(monarbre) then return donnee.operande;
  else
    x := evaluer(gauche(monarbre) );
    y := evaluer(droite(monarbre) );
    return calcul(donnee ,x ,y);
  end if;
end evaluer;

  -- ********************************************************************
  -- *************** procédures publiques au paquetage ***************
  -- ********************************************************************

   -- creer une expression dans le tas
   procedure creer_expression (Ptr_exp : out ptr_expression) is
   begin
     -- allocation dynamique pour ptr_exp
     ptr_exp := new t_expression;
   end creer_expression;

   -- détruire une expression
   procedure free_exp is new ada.unchecked_deallocation(t_expression,
ptr_expression);
   procedure detruire_expression(Ptr_exp : out ptr_expression) is
   begin
       free_exp(ptr_exp);
   end detruire_expression;

   procedure saisir (s : string; ptr_exp : ptr_expression) is
   begin
     put(s);
     get_line(ptr_exp.all.chaine,ptr_exp.all.longueur);
   end saisir;

   function taille (ptr_exp : ptr_expression) return natural is
   begin
    return ptr_exp.all.longueur;
   end taille;

   function evaluer (ptr_exp : ptr_expression) return integer is
    resultat : integer; -- résultat du calcul
   begin
     -- charger les opérateurs et opérandes dans l'arbre
     Charger_Expression(ptr_exp, ptr_exp.arbre'access);

     -- calculer le resultat par parcours de l'arbre binaire
     resultat := evaluer ( ptr_exp.arbre );

     -- libérer la mémoire occupée par l'arbre binaire
     detruire_arbre(ptr_exp.arbre);

     -- renvoyer le résultat
     return resultat;
   end evaluer;

end p_expression;



--------------------------------------------------------------------------------
--  Projet : Evaluation expression arithmétique préfixée
--
--  Nom Module :  pp.adb
--  Numéro de version Module : V 1.0
--  But du Module : évaluer une expression arithmétique préfixée saisie au
--                  clavier et stockée en mémoire sous forme d'arbre binaire.
--
```

```
--                  l'expression devra obligatoirement débuter par une
--                  parenthèse ouvrante et finir par une parenthèse fermante.
--                  donc pas de blanc en début ou en fin d'expression
--          ***** pour simplifier on ne testera pas la validité de l'expression.
--
--                  les opérateurs devront être séparé des opérandes par au
--                  moins un blanc.
--
--                  définition d'un nombre : suite de chiffre sans blanc
--
--                définition d'un nombre négatif : signe moins suivi d'un nombre
--                    (aucun blanc ne sépare le signe du nombre)
--
--  Développeur        /              but de la modif               /  Date
--   C.PIOMBO          /   Création du module                       /  10/05/00
--                     /                                            /
-------------------------------------------------------------------------------
with Ada.Text_Io, Ada.integer_Text_Io; -- E/S Standard
use Ada.Text_Io, Ada.integer_Text_Io;

with p_expression; -- paquetage de manipulation d'une expression arithmétique
use p_expression; -- préfixée

procedure Pp is
--**********************************
--      Les Variables Globales
--**********************************
-- l'expression : on utilise un pointeur pour la mise en oeuvre de la
Expression : ptr_expression; -- récursivité
resultat : integer; -- résultat de l'évaluation. ATTENTION a la division entière

begin
-- ////////////////////    Corps du programme principal    ///////////////////////

   -- ********* saisir l'expression arithmétique préfixée au clavier *********
   creer_expression(expression); -- une expression vide dans le tas
   saisir(" Saisir l'expression arithmetique prefixee ", expression);

   -- si l'expression n'est pas vide
   if taille(expression) > 0 then
      -- ********* évaluation de l'expression arithmétique *********
      resultat := evaluer( expression );
      -- ********* afficher le résultat *********
      put (" resultat du calcul : "); put(resultat);
   else
      Put_Line(" Aucune expression de saisie ");
   end if;

   -- libérer la mémoire occupée dans le tas
   detruire_expression(expression);

exception
   when operateur_inconnu_Error => Put_Line("operateur inconnu dans l'expression
arithmetique");
   when operande_error => put_line(" nombre trop grand ");
end Pp;
```