```
-------------------------------------------------------------------------------
-- NOM DU CSU (principal)           : calcularbrebinaire.adb
-- AUTEUR DU CSU                    : Pascal Pignard
-- VERSION DU CSU                   : 1.0a
-- DATE DE LA DERNIERE MISE A JOUR  : 30 décembre 2012
-- ROLE DU CSU                      : Opérations sur les arbres binaires.
--
--
-- FONCTIONS EXPORTEES DU CSU       :
--
-- FONCTIONS LOCALES DU CSU         :
--
--
-- NOTES                            : Ada 95
--
-- COPYRIGHT                        : (c) Pascal Pignard 2012
-- LICENCE                          : CeCILL V2 (http://www.cecill.info)
-- CONTACT                          : http://blady.pagesperso-orange.fr
-------------------------------------------------------------------------------

with Ada.Unchecked_Deallocation;
with Ada.Text_IO; use Ada.Text_IO;
procedure CalculArbreBinaire is
   generic
      -- Type de la clef de tri
      type TClef is private;
      -- Type des éléments triés suivant la clef
      type TElement is private;
      -- Élement non défini indiquant notamment une recherche non aboutie
      NonDefini : TElement;
      -- Pour l'affichage
      with function Image (E : TElement) return String is <>;
      -- Relation d'ordre de la clef de tri
      with function "<" (Left, Right : TClef) return Boolean is <>;

   package ArbMgr is
      type Arbre is private;

      -- Ajoute un élément à l'arbre binaire en le triant par l'ordre défini par la clef
      -- si la clef existe déjà alors l'élément est remplacé
      procedure Ajoute (A : in out Arbre; Clef : TClef; Element : TElement);
      -- Retire un élément appartenant à l'arbre binaire en gardant l'ordre défini par la cle
      procedure Retire (A : in out Arbre; Clef : TClef; Element : out TElement);
      -- Procédure qui balance l'arbre de façon à équilibrer ces branches filles
      procedure Balance (A : in out Arbre);
      -- Fonction qui retournant l'équibrage de l'arbre à une incertitude près
      function Est_Equilibré (A : Arbre; Incertitude : Natural := 1) return Boolean;
      -- Procédure qui recherche un élément dans l'arbre binaire et qui renvoie son Element
      procedure Recherche (A : Arbre; Clef : TClef; Element : out TElement);
      -- Renvoir l'élément à la racinne de l'arbre
      function Racine (A : Arbre) return TElement;
      -- Renvoie le nombre d'arbre équivalents (2n)!/(N+1)/(n!)^2
      function Nb_Arbres_Equivalents (A : Arbre) return Positive;
      -- Renvoie le nombre de noeuds (càd le nombre d'éléments)
      function Nb_Noeuds (A : Arbre) return Natural;
      -- Renvoie le nombre de noeuds terminaux (sans branches filles ni inférieures ni supéri
      function Nb_Feuilles (A : Arbre) return Natural;
      -- Renvoie le nombr de niveaux
      function Hauteur (A : Arbre) return Natural;
      -- Distance de la clef à la racine
      function Profondeur (A : Arbre; Clef : TClef) return Natural;
```

```
      -- Renvoie l'élément pour lequel la clef est minimale
      function Minimum (A : Arbre) return TElement;
      -- Renvoie l'élément pour lequel la clef est maximale
      function Maximum (A : Arbre) return TElement;
      -- Revoie l'élément correspondant au prédécesseur de la clef
      function Prédécesseur (A : Arbre; Clef : TClef) return TElement;
      -- Revoie l'élément correspondant au successeur de la clef
      function Successeur (A : Arbre; Clef : TClef) return TElement;
      -- Procédure d'affichage de l'arbre binaire
      procedure Affiche (A : Arbre);
      -- Procédure de destruction de l'arbre binaire
      procedure Vide (A : in out Arbre);

   private
      -- Définition d'un noeud pour la gestion de l'arbre binaire.
      type TNoeud;
      type Arbre is access TNoeud;
      subtype PNoeud is Arbre;
      type TNoeud is record
         Gauche  : PNoeud;       -- branche inférieure de l'arbre
         Droit   : PNoeud;       -- branche supérieure de l'arbre
         Clef    : TClef;        -- clef de comparaison
         Element : TElement;     -- stockage de l'élément à trier ou à rechercher
      end record;
   end ArbMgr;

   package body ArbMgr is

      procedure Ajoute (A : in out Arbre; Clef : TClef; Element : TElement) is
         NoeudNouveau : PNoeud;
         procedure AjouteDans (Noeud : PNoeud) is
         begin
            if Clef /= Noeud.Clef then
               if Clef < Noeud.Clef then
                  if Noeud.Gauche /= null then
                     AjouteDans (Noeud.Gauche);
                  else
                     Noeud.Gauche := NoeudNouveau;
                  end if;
               else
                  if Noeud.Droit /= null then
                     AjouteDans (Noeud.Droit);
                  else
                     Noeud.Droit := NoeudNouveau;
                  end if;
               end if;
            else
               Noeud.Element := Element;
            end if;
         end AjouteDans;
      begin
         NoeudNouveau :=
           new TNoeud'(Gauche => null, Droit => null, Clef => Clef, Element => Element);
         if A /= null then
            AjouteDans (A);
         else
            A := NoeudNouveau;
         end if;
      end Ajoute;

      procedure Retire (A : in out Arbre; Clef : TClef; Element : out TElement) is
```

```
      Récup : PNoeud := null;
      procedure RetireDans (Noeud : in out PNoeud) is
         procedure Free is new Ada.Unchecked_Deallocation (TNoeud, PNoeud);
         procedure Insère (De : PNoeud; Vers : in out PNoeud) is
            procedure InsèreDans (Noeud : PNoeud) is
            begin
               if De.Clef = Noeud.Clef then
                  raise Program_Error;
               elsif De.Clef < Noeud.Clef then
                  if Noeud.Gauche /= null then
                     InsèreDans (Noeud.Gauche);
                  else
                     Noeud.Gauche := De;
                  end if;
               elsif Noeud.Droit /= null then
                  InsèreDans (Noeud.Droit);
               else
                  Noeud.Droit := De;
               end if;
            end InsèreDans;
         begin
            if Vers = null then
               Vers := De;
            elsif De /= null then
               InsèreDans (Vers);
            end if;
         end Insère;
      begin
         if Clef = Noeud.Clef then
            Element := Noeud.Element;
            Insère (Noeud.Gauche, Noeud.Droit);
            Récup := Noeud.Droit;
            Free (Noeud);
         elsif Clef < Noeud.Clef then
            if Noeud.Gauche /= null then
               RetireDans (Noeud.Gauche);
               if Récup /= null then
                  Noeud.Gauche := Récup;
                  Récup        := null;
               end if;
            end if;
         elsif Noeud.Droit /= null then
            RetireDans (Noeud.Droit);
            if Récup /= null then
               Noeud.Droit := Récup;
               Récup       := null;
            end if;
         end if;
      end RetireDans;
   begin
      Element := NonDefini;
      if A /= null then
         RetireDans (A);
      end if;
   end Retire;

   procedure Balance (A : in out Arbre) is
      -- Définition du tableau pour le ré-équilibrage de l'arbre
      type TTab is array (Positive range <>) of PNoeud;
      type PTab is access TTab;
      procedure Free is new Ada.Unchecked_Deallocation (TTab, PTab);
```

```
      Tab : PTab := null;

      procedure PlaceDansTab (Noeud : PNoeud) is
         Ancien : PTab;
      begin
         if Noeud.Gauche /= null then
            PlaceDansTab (Noeud.Gauche);
         end if;
         if Tab = null then
            Tab := new TTab'(Positive'First => Noeud); -- premier élément du tableau
         else
            Ancien := Tab;
            Tab    := new TTab'(Tab (Tab'Range) & Noeud); -- les suivants
            Free (Ancien);
         end if;
         if Noeud.Droit /= null then
            PlaceDansTab (Noeud.Droit);
         end if;
      end PlaceDansTab;

      procedure PlaceDansArbre (Noeud : out PNoeud; Premier, Dernier : Positive) is
         Index : Positive;
      begin
         Index := (Premier + Dernier) / 2;
         Noeud := Tab (Index);
         if Premier /= Index then
            PlaceDansArbre (Noeud.Gauche, Premier, Index - 1);
         else
            Noeud.Gauche := null;
         end if;
         if Dernier /= Index then
            PlaceDansArbre (Noeud.Droit, Index + 1, Dernier);
         else
            Noeud.Droit := null;
         end if;
      end PlaceDansArbre;

   begin
      PlaceDansTab (A);
      PlaceDansArbre (A, Tab'First, Tab'Last);
      Free (Tab);
   end Balance;

   function Est_Equilibré (A : Arbre; Incertitude : Natural := 1) return Boolean is
   begin
      return abs (Hauteur (A.Gauche) - Hauteur (A.Droit)) <= Incertitude;
   end Est_Equilibré;

   procedure Recherche (A : Arbre; Clef : TClef; Element : out TElement) is
      procedure RechercheDans (Noeud : PNoeud) is
      begin
         if Clef = Noeud.Clef then
            Element := Noeud.Element;
         elsif Clef < Noeud.Clef then
            if Noeud.Gauche /= null then
               RechercheDans (Noeud.Gauche);
            end if;
         elsif Noeud.Droit /= null then
            RechercheDans (Noeud.Droit);
         end if;
      end RechercheDans;
```

```
begin
   Element := NonDefini;
   if A /= null then
      RechercheDans (A);
   end if;
end Recherche;

function Racine (A : Arbre) return TElement is
begin
   if A /= null then
      return A.Element;
   else
      return NonDefini;
   end if;
end Racine;

function Nb_Arbres_Equivalents (A : Arbre) return Positive is
   function Produit (De, A : Natural) return Long_Long_Integer is
      F : Long_Long_Integer := 1;
   begin
      for Ind in De .. A loop
         F := F * Long_Long_Integer (Ind);
      end loop;
      return F;
   end Produit;
   N : constant Natural := Nb_Noeuds (A); -- ok jusqu'à 14 inclus !!
begin
   -- (2n)! / (n+1) / (n!)^2 réduit à ∏ des k=n+2..2n / n!
   return Positive (Produit (N + 2, 2 * N) / Produit (1, N));
end Nb_Arbres_Equivalents;

function Nb_Noeuds (A : Arbre) return Natural is
begin
   if A /= null then
      return Nb_Noeuds (A.Gauche) + 1 + Nb_Noeuds (A.Droit);
   else
      return 0;
   end if;
end Nb_Noeuds;

function Nb_Feuilles (A : Arbre) return Natural is
begin
   if A = null then
      return 0;
   elsif A.Gauche = null and then A.Droit = null then
      return 1;
   else
      return Nb_Feuilles (A.Gauche) + Nb_Feuilles (A.Droit);
   end if;
end Nb_Feuilles;

function Hauteur (A : Arbre) return Natural is
begin
   if A /= null then
      return 1 + Natural'Max (Hauteur (A.Gauche), Hauteur (A.Droit));
   else
      return 0;
   end if;
end Hauteur;

function Profondeur (A : Arbre; Clef : TClef) return Natural is
```

```
      Prof : Natural := 0;
      procedure ProfondeurDans (Noeud : PNoeud) is
      begin
         if Clef = Noeud.Clef then
            return;
         elsif Clef < Noeud.Clef then
            if Noeud.Gauche /= null then
               Prof := Prof + 1;
               ProfondeurDans (Noeud.Gauche);
            end if;
         elsif Noeud.Droit /= null then
            Prof := Prof + 1;
            ProfondeurDans (Noeud.Droit);
         end if;
      end ProfondeurDans;
   begin
      if A /= null then
         ProfondeurDans (A);
      end if;
      return Prof;
   end Profondeur;

   function Minimum (A : Arbre) return TElement is
   begin
      if A /= null then
         if A.Gauche /= null then
            return Minimum (A.Gauche);
         else
            return A.Element;
         end if;
      else
         return NonDefini;
      end if;
   end Minimum;

   function Maximum (A : Arbre) return TElement is
   begin
      if A /= null then
         if A.Droit /= null then
            return Maximum (A.Droit);
         else
            return A.Element;
         end if;
      else
         return NonDefini;
      end if;
   end Maximum;

   function Prédécesseur (A : Arbre; Clef : TClef) return TElement is
      LPred : TElement := NonDefini;
      procedure Pred (Noeud : PNoeud) is
      begin
         if Noeud /= null then
            if Noeud.Clef = Clef then
               if Noeud.Gauche /= null then
                  LPred := Maximum (Noeud.Gauche);
               end if;
            elsif Noeud.Clef < Clef then
               LPred := Noeud.Element;
               Pred (Noeud.Droit);
            else
```

```
            Pred (Noeud.Gauche);
         end if;
      else
         return;
      end if;
   end Pred;
begin
   Pred (A);
   return LPred;
end Prédécesseur;

function Successeur (A : Arbre; Clef : TClef) return TElement is
   LSucc : TElement := NonDefini;
   procedure Succ (Noeud : PNoeud) is
   begin
      if Noeud /= null then
         if Noeud.Clef = Clef then
            if Noeud.Droit /= null then
               LSucc := Minimum (Noeud.Droit);
            end if;
         elsif Noeud.Clef < Clef then
            Succ (Noeud.Droit);
         else
            LSucc := Noeud.Element;
            Succ (Noeud.Gauche);
         end if;
      else
         return;
      end if;
   end Succ;
begin
   Succ (A);
   return LSucc;
end Successeur;

procedure Affiche (A : Arbre) is
   procedure Affiche (Noeud : PNoeud; Pos : Integer) is
   begin
      Put (Image (Noeud.Element));
      if Noeud.Droit /= null then
         Put ("--");
         Affiche (Noeud.Droit, Pos + 1);
      end if;
      if Noeud.Gauche /= null then
         New_Line;
         for Tab in 1 .. Pos loop
            Put (" |  ");
         end loop;
         New_Line;
         for Tab in 1 .. Pos - 1 loop
            Put (" |  ");
         end loop;
         Affiche (Noeud.Gauche, Pos);
      end if;
   end Affiche;
begin
   if A /= null then
      Affiche (A, 1);
   end if;
   New_Line;
end Affiche;
```

```
      procedure Vide (A : in out Arbre) is
         procedure Elimine (Noeud : in out PNoeud) is
            procedure Free is new Ada.Unchecked_Deallocation (TNoeud, PNoeud);
         begin
            if Noeud.Gauche /= null then
               Elimine (Noeud.Gauche);
            end if;
            if Noeud.Droit /= null then
               Elimine (Noeud.Droit);
            end if;
            Free (Noeud);
         end Elimine;
      begin
         if A /= null then
            Elimine (A);
         end if;
      end Vide;

   end ArbMgr;

   subtype TClef is String (1 .. 10);
   subtype TElement is Integer;
   NonDefini : constant TElement := 0;
   Element   : TElement;

   function Image (E : TElement) return String is
   begin
      return E'Img;
   end Image;

   -- Création des arbres
   package MonArbre is new ArbMgr (TClef, TElement, NonDefini);
   use MonArbre;
   A1, A2 : MonArbre.Arbre;

   function Format (Ind : TElement) return TClef is
      Dum   : constant String := TElement'Image (Ind);
      Blanc : constant TClef  := "          ";
   begin
      return Blanc (1 .. 10 - Dum'Last) & Dum;
   end Format;

   subtype Interval is TElement range 1 .. 100;
   Tab : constant array (1 .. 10) of TElement := (5, 2, 9, 4, 1, 3, 8, 6, 7, 10);
begin
   Put_Line ("Essai de l'Unité : ArbreMgr.");

   Put_Line ("Création de l'arbre A1.");
   for Ind in Interval loop
      MonArbre.Ajoute (A1, Format (Ind * 3), Ind * 3);
   end loop;
   Put_Line ("Nombre d'éléments :" & Format (Interval'Last));

   MonArbre.Recherche (A1, Format (Interval'First * 3), Element);
   Put_Line ("Élement recherché premier :" & TElement'Image (Element));
   MonArbre.Recherche (A1, Format (Interval'Last * 3), Element);
   Put_Line ("Élement recherché dernier :" & TElement'Image (Element));
   Put_Line ("Élement racine :" & TElement'Image (MonArbre.Racine (A1)));
   Put_Line ("Nombre de noeuds :" & Natural'Image (MonArbre.Nb_Noeuds (A1)));
   Put_Line ("Nombre de feuilles :" & Natural'Image (MonArbre.Nb_Feuilles (A1)));
```

```
   Put_Line ("Hauteur :" & Natural'Image (MonArbre.Hauteur (A1)));
   Put_Line ("Profondeur de 30 :" & Natural'Image (MonArbre.Profondeur (A1, Format (30))));
   Put_Line ("Élement minimum :" & TElement'Image (MonArbre.Minimum (A1)));
   Put_Line ("Élement maximum :" & TElement'Image (MonArbre.Maximum (A1)));
   MonArbre.Retire (A1, Format (201), Element);
   Put_Line ("Élement retiré 201 :" & TElement'Image (Element));
   Put_Line
     ("Élement prédécesseur de 201 :" &
      TElement'Image (MonArbre.Prédécesseur (A1, Format (201))));
   Put_Line
     ("Élement successeur de 201 :" & TElement'Image (MonArbre.Successeur (A1, Format (201)))

   Put_Line ("Est équilibré :" & Boolean'Image (MonArbre.Est_Equilibré (A1)));
   Put_Line ("Appel à 'Balance'.");
   MonArbre.Balance (A1);
   Put_Line ("Est équilibré :" & Boolean'Image (MonArbre.Est_Equilibré (A1)));

   MonArbre.Recherche (A1, Format (Interval'First * 3), Element);
   Put_Line ("Élement recherché premier :" & TElement'Image (Element));
   MonArbre.Recherche (A1, Format (Interval'Last * 3), Element);
   Put_Line ("Élement recherché dernier :" & TElement'Image (Element));
   Put_Line ("Élement racine :" & TElement'Image (MonArbre.Racine (A1)));
   Put_Line ("Nombre de noeuds :" & Natural'Image (MonArbre.Nb_Noeuds (A1)));
   Put_Line ("Nombre de feuilles :" & Natural'Image (MonArbre.Nb_Feuilles (A1)));
   Put_Line ("Hauteur :" & Natural'Image (MonArbre.Hauteur (A1)));
   Put_Line ("Profondeur de 30 :" & Natural'Image (MonArbre.Profondeur (A1, Format (30))));
   Put_Line ("Élement minimum :" & TElement'Image (MonArbre.Minimum (A1)));
   Put_Line ("Élement maximum :" & TElement'Image (MonArbre.Maximum (A1)));
   MonArbre.Retire (A1, Format (204), Element);
   Put_Line ("Élement retiré 204 :" & TElement'Image (Element));
   Put_Line
     ("Élement prédécesseur de 201 :" &
      TElement'Image (MonArbre.Prédécesseur (A1, Format (201))));
   Put_Line
     ("Élement successeur de 201 :" & TElement'Image (MonArbre.Successeur (A1, Format (201)))

   MonArbre.Vide (A1);
   Put_Line ("Arbre vidé.");

   Put_Line ("Création de l'arbre A2.");
   for Ind in Tab'Range loop
      MonArbre.Ajoute (A2, Format (Tab (Ind)), Tab (Ind));
   end loop;
   Put_Line
     ("Nombre d'arbres équivalents :" & Natural'Image (MonArbre.Nb_Arbres_Equivalents (A2)));
   MonArbre.Affiche (A2);
   Put_Line ("Appel à 'Balance'.");
   MonArbre.Balance (A2);
   MonArbre.Affiche (A2);
   Put_Line ("Élimination de l'élément 8.");
   MonArbre.Retire (A2, Format (8), Element);
   MonArbre.Affiche (A2);
   MonArbre.Vide (A2);
   Put_Line ("Arbre vidé.");

end CalculArbreBinaire;
```