```
--------------------------------------------------------------------------------
-- NOM DU CSU (principal)           : calculdelistes.adb
-- AUTEUR DU CSU                    : Pascal Pignard
-- VERSION DU CSU                   : 1.0a
-- DATE DE LA DERNIERE MISE A JOUR  : 17 mai 2012
-- ROLE DU CSU                      : Opérations sur les listes.
--
--
-- FONCTIONS EXPORTEES DU CSU       :
--
-- FONCTIONS LOCALES DU CSU         :
--
--
-- NOTES                            :
--
-- COPYRIGHT                        : (c) Pascal Pignard 2012
-- LICENCE                          : CeCILL V2 (http://www.cecill.info)
-- CONTACT                          : http://blady.pagesperso-orange.fr
--------------------------------------------------------------------------------

with Ada.Containers.Doubly_Linked_Lists;
with Ada.Float_Text_IO; use Ada.Float_Text_IO;

procedure CalculDeListes is
   generic
      type TElement is private;
   package Listes is
      type Liste is tagged private;
      Liste_Vide : constant Liste;
      type Tableau is array (Positive range <>) of TElement;
      function Créé_Liste (T : Tableau) return Liste;
      function "&" (Gauche, Droite : Liste) return Liste;
      function "&" (Gauche : Liste; Droite : TElement) return Liste;
      function "&" (Gauche : TElement; Droite : Liste) return Liste;
      function Est_Vide (L : Liste) return Boolean;
      function Longueur (L : Liste) return Natural;
      function Tête (L : Liste) return TElement;
      function Queue (L : Liste) return TElement;
      function Position (L : Liste; P : Positive) return TElement;
      function Renverse (L : Liste) return Liste;
      function Applique
        (L    : Liste;
         F    : access function (E : TElement) return TElement)
         return Liste;
      procedure Scinde (L : Liste; Tête : out TElement; Reste : out Liste);
   private
      package IntListes is new Ada.Containers.Doubly_Linked_Lists (TElement);
      type Liste is new IntListes.List with null record;
      Liste_Vide : constant Liste := (IntListes.Empty_List with null record);
   end Listes;

   package body Listes is

      function Créé_Liste (T : Tableau) return Liste is
      begin
         return L : Liste do
            for Index in T'Range loop
               Append (L, T (Index));
            end loop;
         end return;
      end Créé_Liste;
```

```
function "&" (Gauche, Droite : Liste) return Liste is
   Curseur : IntListes.Cursor := First (Droite);
   use type IntListes.Cursor;
begin
   return L : Liste := Gauche do
      while Curseur /= IntListes.No_Element loop
         Append (L, IntListes.Element (Curseur));
         IntListes.Next (Curseur);
      end loop;
   end return;
end "&";

function "&" (Gauche : Liste; Droite : TElement) return Liste is
begin
   return L : Liste := Gauche do
      Append (L, Droite);
   end return;
end "&";

function "&" (Gauche : TElement; Droite : Liste) return Liste is
begin
   return L : Liste := Droite do
      Prepend (L, Gauche);
   end return;
end "&";

function Est_Vide (L : Liste) return Boolean renames Is_Empty;

function Longueur (L : Liste) return Natural is
begin
   return Natural (Length (L));
end Longueur;

function Tête (L : Liste) return TElement renames First_Element;

function Queue (L : Liste) return TElement renames Last_Element;

function Position (L : Liste; P : Positive) return TElement is
   Curseur : IntListes.Cursor := First (L);
begin
   for Ind in 2 .. P loop
      IntListes.Next (Curseur);
   end loop;
   return IntListes.Element (Curseur);
end Position;

function Renverse (L : Liste) return Liste is
begin
   return RL : Liste := L do
      Reverse_Elements (RL);
   end return;
end Renverse;

function Applique
  (L    : Liste;
   F    : access function (E : TElement) return TElement)
   return Liste
is
   Curseur : IntListes.Cursor := First (L);
   use type IntListes.Cursor;
```

```
      begin
         return RL : Liste do
            while Curseur /= IntListes.No_Element loop
               Append (RL, F (IntListes.Element (Curseur)));
               IntListes.Next (Curseur);
            end loop;
         end return;
      end Applique;

      procedure Scinde (L : Liste; Tête : out TElement; Reste : out Liste) is
         Curseur : IntListes.Cursor := First (L);
         use type IntListes.Cursor;
      begin
         if Curseur /= IntListes.No_Element then
            Tête  := IntListes.Element (Curseur);
            IntListes.Next (Curseur);
            while Curseur /= IntListes.No_Element loop
               Append (Reste, IntListes.Element (Curseur));
               IntListes.Next (Curseur);
            end loop;
         else
            raise Constraint_Error;
         end if;
      end Scinde;

   end Listes;

   package ListesRéelles is new Listes (Float);
   use ListesRéelles;

   function Mini (L : Liste) return Float is
      M, A : Float;
      LP   : Liste;
   begin
      if L.Longueur = 1 then
         return L.Tête;
      else
         L.Scinde (A, LP);
         M := Mini (LP);
         if M < A then
            return M;
         else
            return A;
         end if;
      end if;
   end Mini;

   function Fois_2 (X : Float) return Float is
   begin
      return X * 2.0;
   end Fois_2;

begin
   Put (Mini (Créé_Liste ((2.0, 4.0, 3.3)) & 1.0));
   Put
     (Position
         (Applique
             (5.0 & Créé_Liste ((2.0, 4.0, 3.3)) & Créé_Liste ((1 => 8.2)),
              Fois_2'Access),
          5));
end CalculDeListes;
```