38 - Rendre générique une unité non générique

Programmation Concurrente - LI330
Université P. & M. Curie - année scolaire 2013/2014

PrC

Objectif

- Montrer comment on rend générique une unité qui ne l'est pas
- Illustrer certains aspects de la généricité
  - Sa puissance
  - Ses dangers
- Reprise de l'unité de gestion d'arbres binaires d'entiers
  - On gèrera des arbres binaires de «ce que l'on veut»
  - On va déterminer les paramètres génériques formels
  - On va identifier les modifications sur le corps de l'unité
  - On va montrer comment c'est transparent sur le texte du programme de test

Paramètres génériques formels de Arbres_Binaires_Gen

- Le type de la valeur associée à un nœud
  - Hypothèse sur les types: contraints
  - Hypothèse sur les opérations: aucune (ni affectation ni recopie d'égalité)
- Une procédure de recopie du type de valeur associé à un nœud
  - On en a besoin pour programmer l'unité
- Une fonction de comparaison d'égalité du type de valeur associé à un nœud
  - On en a également besoin pour programmer l'unité

Spécification de Arbres_Binaires_Gen (1)


```
-- Gestionnaire d'arbres binaires générique
generic
   type Un_Contenu is limited private;
   with procedure Copie (A : in Un_Contenu;
                         B : out Un_Contenu);
   with function "=" (A, B : Un_Contenu) return Boolean is <>;
package Arbres_Binaires_Gen is

   -- Des exceptions utiles
   Arbre_Vide_Error : exception;
   Element_Courant_Null_Error : exception;
   Position_Non_Vide_Error : exception;
   Position_Vide_Error : exception;

   -- Un type pour définir si on est à gauche ou à droite de l'arbre
   type Direction is (Gauche, Droite);
   -- le type arbre binaire (limité-privé)
   type Un_Arbre_Binaire is limited private;
```

Spécification de Arbres_Binaires_Gen (2)


```
-- Créer un arbre binaire vide
procedure Creer (A : out Un_Arbre_Binaire);
-- Détruire un arbre binaire
procedure Detruire (A : in out Un_Arbre_Binaire);

-- Récupérer le nombre d'éléments d'un arbre binaire
function Nombre_D_Elements (A : in Un_Arbre_Binaire)
   return Natural;
-- Récupérer la profondeur d'un arbre binaire
function Profondeur (A : Un_Arbre_Binaire) return Natural;
-- Récupérer la profondeur minimale d'un arbre binaire
function Profondeur_Min (A : Un_Arbre_Binaire) return Natural;
-- Se déplacer dans l'arbre
-- raise Arbre_Vide_Error
procedure Aller_Racine (A : in out Un_Arbre_Binaire);
-- raise Element_Courant_Null_Error
procedure Descendre (A : in out Un_Arbre_Binaire;
                     D : in Direction);
```

Spécification de Arbres_Binaires_Gen (3)


```
-- Savoir où on se trouve
-- raise Element_Courant_Null_Error
function Element_Courant_Est_Feuille (A : Un_Arbre_Binaire)
   return Boolean;
-- raise Element_Courant_Null_Error
function Element_Courant_Est_Noeud (A : Un_Arbre_Binaire)
   return Boolean;
-- raise Element_Courant_Null_Error
function Element_Courant_Est_Vide (A : Un_Arbre_Binaire)
   return Boolean;
-- Récupérer la valeur de l'élément courant
-- raise Element_Courant_Null_Error
function Etiquette_Element_Courant (A : in Un_Arbre_Binaire)
   return Un_Contenu;
-- Ajouter un élément à une direction donnée que l'on suppose vide
-- dans l'élément courant
-- raise Position_Non_Vide_Error
procedure Ajoute_Noeud (V : in Un_Contenu;
                        D : in Direction;
                        A : in out Un_Arbre_Binaire);
```

Spécification de Arbres_Binaires_Gen (4)


```
-- Supprimer un sous-arbre à partir de l'élément courant dans une direction donnée
-- raise Position_Vide_Error
-- raise Element_Courant_Null_Error
procedure Supprime_Fils (A : in out Un_Arbre_Binaire;
                         D : in Direction);

-- Recopier un arbre
procedure Copie (A1 : in Un_Arbre_Binaire;
                 A2 : out Un_Arbre_Binaire);

-- Comparer deux arbres
function "=" (A1, A2 : Un_Arbre_Binaire) return Boolean;
```

Spécification de Arbres_Binaires_Gen (5)


```
private
   type Noeud; -- Déclaration du type noeud

   type A_Noeud is access Noeud; -- Déclaration du pointeur sur un noeud

   type Tab_Fils is array (Direction) of A_Noeud; -- Tableau de pointeurs

   type Noeud is record -- Structure d'un noeud
      Contenu : Un_Contenu;
      Descendance : Tab_Fils := (others => null);
   end record;

   type Un_Arbre_Binaire is record -- Description d'un arbre
      Nombre_D_Elements : Natural := 0;
      Element_Courant : A_Noeud := null;
      Racine : A_Noeud := null;
   end record;

end Arbres_Binaires_Gen;
```

Corps de Arbres_Binaires_Gen (1)

```
with Unchecked_deallocation;
with Ada.Text_Io; use Ada.Text_Io;
package body Arbres_Binaires_Gen is
   -- Création de la procédure de désallocation
   procedure Desalloue_Noeud is new
        Unchecked_deallocation (Noeud, A_Noeud);
   -- Procedure privée (n'apparait pas dans la spécification de l'unité)
   procedure Desalloue_Recursivement (An : in out A_Noeud;
                                      Nb_El : out Natural) is
      N1, N2 : Natural;
   begin
      if An /= null then
         Desalloue_Recursivement (An.Descendance (Gauche), N1);
         Desalloue_Recursivement (An.Descendance (Droite), N2);
         Desalloue_Noeud (An);
         Nb_El := N1 + N2 + 1;
      else
         Nb_El := 0;
      end if;
   end Desalloue_Recursivement;
```

Corps de Arbres_Binaires_Gen (2)


```
procedure Creer (A : out Un_Arbre_Binaire) is
begin
   A := (0, null, null);
end Creer;

procedure Detruire (A : in out Un_Arbre_Binaire) is
   Nb_El : Natural;
begin
   Desalloue_Recursivement (A.Racine, Nb_El);
   A := (0, null, null);
end Detruire;

function Nombre_D_Elements (A : Un_Arbre_Binaire)
   return Natural is

begin
   return A.Nombre_D_Elements;
end Nombre_D_Elements;
```

Corps de Arbres_Binaires_Gen (3)


```
function Profondeur (A : Un_Arbre_Binaire) return Natural is

   function Calcule_Profondeur (An : A_Noeud) return Natural is

   begin
      if An = null then
         return 0;
      end if;
      return Natural'Max (Calcule_Profondeur
                                (An.Descendance (Gauche)),
                          Calcule_Profondeur
                                (An.Descendance (Droite))) + 1;
   end Calcule_Profondeur;

begin -- Profondeur
   return Calcule_Profondeur (A.Racine);
end Profondeur;
```

Corps de Arbres_Binaires_Gen (4)


```
function Profondeur_Min (A : in Un_Arbre_Binaire)
   return Natural is
   Escape : exception ;
   function Prof_Min (An : A_Noeud;
                      Min_Connu : Natural) return Natural is
   -- raise Escape lorsqu'il n'existe pas de noeud de profondeur < Min_Connu
   -- rend la profondeur minimale de l'arbre de racine An
      Prof  : Natural;
   begin
      if An = null then
         return 0;
      end if;
      if Min_Connu = 1 then
         raise Escape ;
      end if;
      begin
         Prof := Prof_Min (An.Descendance (Gauche), Min_Connu -1);
      exception
         when Escape =>
            return Prof_Min (An.Descendance (Droite),
                             Min_Connu -1) + 1;
             -- Noter si Prof_Min lève l'exception Escape, elle sera propagée
      end;
```

```
      begin
         return Prof_Min (An.Descendance (Droite), Prof) + 1 ;
      exception
         when Escape =>
            return Prof + 1;
      end ;
   end Prof_Min;

begin -- Profondeur_Min
   return Prof_Min(A.Racine, Natural'Last) ;
end Profondeur_Min;

 -- raise Arbre_Vide_Error
 procedure Aller_Racine (A : in out Un_Arbre_Binaire) is

 begin
    if A.Racine = null then
       raise Arbre_Vide_Error;
    end if;
    A.Element_Courant := A.Racine;
 end Aller_Racine;
```

Corps de Arbres_Binaires_Gen (6)

```
-- raise Element_Courant_Null_Error
procedure Descendre (A : in out Un_Arbre_Binaire;
                     D : in Direction) is
begin
   A.Element_Courant := A.Element_Courant.Descendance(D);
exception
   when Constraint_Error =>
      raise Element_Courant_Null_Error;
end Descendre;

-- raise Element_Courant_Null_Error
function Element_Courant_Est_Feuille (A : Un_Arbre_Binaire)
   return Boolean is

begin
   return A.Element_Courant.Descendance(Gauche) = null and
A.Element_Courant.Descendance(Droite) = null;
exception
   when Constraint_Error =>
      raise Element_Courant_Null_Error;
end Element_Courant_Est_Feuille;
```

Corps de Arbres_Binaires_Gen (7)

```
-- raise Element_Courant_Null_Error
function Element_Courant_Est_Noeud (A : Un_Arbre_Binaire)
   return Boolean is

begin
   return A.Element_Courant.Descendance(Gauche) /= null or
          A.Element_Courant.Descendance(Droite) /= null;
exception
   when Constraint_Error =>
      raise Element_Courant_Null_Error;
end Element_Courant_Est_Noeud;

function Element_Courant_Est_Vide (A : Un_Arbre_Binaire)
   return Boolean is

begin
   return A.Element_Courant = null;
end Element_Courant_Est_Vide;
```

Corps de Arbres_Binaires_Gen (8)


```
-- raise Element_Courant_Null_Error
function Etiquette_Element_Courant (A : Un_Arbre_Binaire)
   return Un_Contenu is

begin
   return A.Element_Courant.all.Contenu;
exception
   when Constraint_Error =>
      raise Element_Courant_Null_Error;
end Etiquette_Element_Courant;
```

Corps de Arbres_Binaires_Gen (9)


```
-- raise Position_Non_Vide_Error
procedure Ajoute_Noeud (V : in Un_Contenu;
                        D : in Direction;
                        A : in out Un_Arbre_Binaire) is
begin
    if A.Nombre_D_Elements = 0 then
       A.Racine := new Noeud;
       Copie (V, A.Racine.all.Contenu);
       A.Racine.Descendance := (others => null);
       A.Element_Courant := A.Racine;
    else
       if A.Element_Courant.Descendance (D) /= null then
          raise Position_Non_Vide_Error;
       end if;
       A.Element_Courant.Descendance (D) := new Noeud;
       Copie (V, A.Element_Courant.all.Descendance (D).all.Contenu);
       A.Element_Courant.Descendance (D).Descendance :=
            (others => null);
    end if;
    A.Nombre_D_Elements := A.Nombre_D_Elements + 1;
end Ajoute_Noeud;
```

Corps de Arbres_Binaires_Gen (10)


```
-- raise Position_Vide_Error
-- raise Element_Courant_Null_Error
procedure Supprime_Fils (A : in out Un_Arbre_Binaire;
                         D : in Direction) is
   Nb_El : Natural;
begin
    if A.Element_Courant.Descendance (D) = null then
       raise Position_Vide_Error;
    end if;
    Desalloue_Recursivement (A.Element_Courant.Descendance (D),
                             Nb_El);
    A.Nombre_D_Elements := A.Nombre_D_Elements - Nb_El;
exception
   when Constraint_Error =>
      raise Element_Courant_Null_Error;
end Supprime_Fils;
```

Corps de Arbres_Binaires_Gen (11)


```
procedure Copie (A1 : in Un_Arbre_Binaire;
                 A2 : out Un_Arbre_Binaire) is
   function Copie_Recursive (An : A_Noeud) return A_Noeud is
      V_Ret : A_Noeud;
   begin
      if An = null then
         return null;
      else
         V_Ret := new Noeud;
         Copie (An.Contenu, V_Ret.Contenu);
         V_Ret.Descendance (Gauche) :=
            Copie_Recursive (An.Descendance (Gauche));
         V_Ret.Descendance (Droite) :=
            Copie_Recursive (An.Descendance (Droite));
         return V_Ret;
      end if;
   end Copie_Recursive;
begin -- Copie
   A2.Nombre_D_Elements := A1.Nombre_D_Elements;
   A2.Racine := Copie_Recursive (A1.Racine);
   A2.Element_Courant := A2.Racine;
end Copie;
```

Corps de Arbres_Binaires_Gen (12)


```
function "=" (A1, A2 : Un_Arbre_Binaire) return Boolean is
   function Comparaison_Recursive (An1, An2 : A_Noeud)
      return Boolean is
   begin
      if An1 = null or An2 = null then
         return An1 = null and An2 = null;
      else
         return An1.all.Contenu = An2.all.Contenu and then
                Comparaison_Recursive (An1.all.Descendance (Gauche),
                   An2.Descendance (Gauche)) and then
                Comparaison_Recursive (An1.all.Descendance (Droite),
                   An2.Descendance (Droite));
      end if;
   end Comparaison_Recursive;
begin -- "="
   if A1.Nombre_D_Elements /= A2.Nombre_D_Elements then
      return False;
   else
      return Comparaison_Recursive (A1.Racine, A2.Racine);
   end if;
end "=";
end Arbres_Binaires_Gen;
```

Une fonction de recopie d'entiers

Nécessaire pour l'instanciation de l'unité générique

```
-- Fonction Copie_Entiers nécessaire pour l'instanciation
-- de l'unité de gestion d'arbres binaires générique

procedure Copie_Entiers (I1 : in Integer;
                         I2 : out Integer) is

begin
   I2 := I1;
end Copie_Entiers;
```

Structure du programme de test

- Sans unité générique

- Avec unité générique

Instanciation de Arbres_Binaires_Gen

```
-- Gestionnaire d'arbres binaires d'entiers obtenu par instanciation
with Arbres_Binaires_Gen, Copie_Entiers;

package Arbres_Binaires_D_Entiers is new Arbres_Binaires_Gen
                (Un_Contenu => Integer,
                 Copie => Copie_Entiers);
```

OU

```
-- Gestionnaire d'arbres binaires d'entiers obtenu par instanciation
with Arbres_Binaires_Gen, Copie_Entiers;

package Arbres_Binaires_D_Entiers is new Arbres_Binaires_Gen
                (Un_Contenu => Integer,
                 Copie => Copie_Entiers,
                 "=" => "=");
```

Test de Arbres_Binaires_Gen (1)


```
with Ada.Text_IO, Arbres_Binaires_D_Entiers, Ada.exceptions;
use Ada.Text_IO, Arbres_Binaires_D_Entiers, Ada.exceptions;

procedure T_Arbres_Binaires_D_Entiers is

   -- Seule ligne modifiée afin de ne pas avoir à changer les déclarations qui suivent.
   -- Il s'agit d'un renommage du type de l'unité instanciée.
   subtype Un_Arbre_Binaire_D_Entiers is
      Arbres_Binaires_D_Entiers.Un_Arbre_Binaire;

A, B : Un_Arbre_Binaire_D_Entiers;

begin
   -- Création d'un arbre
   Creer (A);
   Put_Line ("Nombre d'elements de A     =" &
             Natural'Image (Nombre_D_Elements (A)));
   Put_Line ("profondeur de A            =" &
             Natural'Image (Profondeur (A)));
```

Test de Arbres_Binaires_Gen (2)


```
-- Ajout de quelques éléments
Ajoute_Noeud (1, Gauche, A);
Put_Line ("L'element courant est vide = " &
          Boolean'Image (Element_Courant_Est_Vide (A)));
Put_Line ("Suis-je sur une feuille    = " &
          Boolean'Image (Element_Courant_Est_Feuille (A)));
Put_Line ("Suis-je sur un noeud       = " &
          Boolean'Image (Element_Courant_Est_Noeud (A)));
Ajoute_Noeud (2, Gauche, A);
Ajoute_Noeud (3, Droite, A);
Put_Line ("L'element courant est vide = " &
          Boolean'Image (Element_Courant_Est_Vide (A)));
Put_Line ("Suis-je sur une feuille    = " &
          Boolean'Image (Element_Courant_Est_Feuille (A)));
Put_Line ("Suis-je sur un noeud       = " &
          Boolean'Image (Element_Courant_Est_Noeud (A)));
Descendre (A, Droite);
Ajoute_Noeud (4, Gauche, A);
Ajoute_Noeud (5, Droite, A);
Descendre (A, Droite);
Ajoute_Noeud (6, Droite, A);
```

Test de Arbres_Binaires_Gen (3)


```
Descendre (A, Droite);
Ajoute_Noeud (7, Droite, A);
Aller_Racine (A);
Descendre (A, Gauche);
Ajoute_Noeud (8, Gauche, A);
Descendre (A, Gauche);
Ajoute_Noeud (9, Gauche, A);
begin
   Put_Line ("Tentative d'ajout sans deplacement...");
   Ajoute_Noeud (10, Gauche, A);
exception
   when E : others =>
      Put_Line ("Levee de " & exception_name (E) &
                " dans Ajoute_Noeud");
end;
Descendre (A, Gauche);
Ajoute_Noeud (10, Gauche, A);
Descendre (A, Gauche);
Ajoute_Noeud (11, Gauche, A);
Descendre (A, Gauche);
```

Test de Arbres_Binaires_Gen (4)

```
Put_Line ("L'element courant est vide = " &
          Boolean'Image (Element_Courant_Est_Vide (A)));
Put_Line ("Suis-je sur une feuille    = " &
          Boolean'Image (Element_Courant_Est_Feuille (A)));
Put_Line ("Suis-je sur un noeud       = " &
          Boolean'Image (Element_Courant_Est_Noeud (A)));
Aller_Racine (A);
Descendre (A, Gauche);
Descendre (A, Gauche);
Put_Line ("L'element courant est vide = " &
          Boolean'Image (Element_Courant_Est_Vide (A)));
Put_Line ("Suis-je sur une feuille    = " &
          Boolean'Image (Element_Courant_Est_Feuille (A)));
Put_Line ("Suis-je sur un noeud       = " &
          Boolean'Image (Element_Courant_Est_Noeud (A)));
Put_Line ("L'element courant est      =" &
          Natural'Image (Etiquette_Element_Courant (A)));
Put_Line ("Nombre d'elements de A     =" &
          Natural'Image (Nombre_D_Elements (A)));
Put_Line ("profondeur de A            =" &
          Natural'Image (Profondeur (A)));
```

Test de Arbres_Binaires_Gen (5)


```
Put_Line ("profondeur Minimale de A   =" &
          Natural'Image (Profondeur_Min (A)));
Supprime_Fils (A, Gauche);
Aller_Racine (A);
Put_Line ("L'element courant est      =" &
          Natural'Image (Etiquette_Element_Courant (A)));
Put_Line ("Nombre d'elements de A     =" &
          Natural'Image (Nombre_D_Elements (A)));
Put_Line ("profondeur de A            =" &
          Natural'Image (Profondeur (A)));
Put_Line ("profondeur Minimale de A   =" &
          Natural'Image (Profondeur_Min (A)));
Copie (A, B);
Put_Line ("Nombre d'elements de B     =" &
          Natural'Image (Nombre_D_Elements (B)));
Put_Line ("profondeur de B            =" &
          Natural'Image (Profondeur (B)));
Put_Line ("profondeur Minimale de B   =" &
          Natural'Image (Profondeur_Min (B)));
Put_Line ("A et B sont-ils egaux      = " &
          Boolean'Image (A = B));
```

Test de Arbres_Binaires_Gen (6)


```
Supprime_Fils (B, Gauche);
Put_Line ("A et B sont-ils egaux      = " &
          Boolean'Image (A = B));
Detruire (A);
Put_Line ("A et B sont-ils egaux      = " &
          Boolean'Image (A = B));
Detruire (B);
Put_Line ("A et B sont-ils egaux      = " &
          Boolean'Image (A = B));
begin
   Put_Line ("Sur un arbre vide...");
   Put_Line ("L'element courant est      =" &
             Natural'Image (Etiquette_Element_Courant (A)));
exception
   when E : others =>
      Put_Line ("Levee de " & exception_name (E) &
                " dans Etiquette_Element_Courant");
end;
```

Test de Arbres_Binaires_Gen (7)


```
   begin
      Put_Line ("Suis-je sur une feuille    = " &
                Boolean'Image (Element_Courant_Est_Feuille (A)));
   exception
      when E : others =>
         Put_Line ("Levee de " & exception_name (E) &
                   " dans Element_Courant_Est_Feuille");
   end;
   begin
      Put_Line ("Suis-je sur un noeud       = " &
                Boolean'Image (Element_Courant_Est_Noeud (A)));
   exception
      when E : others =>
         Put_Line ("Levee de " & exception_name (E) &
                   " dans Element_Courant_Est_Noeud");
   end;
   begin
      Supprime_Fils (A, Gauche);
   exception
      when E : others =>
         Put_Line ("Levee de " & exception_name (E) &
                   " dans Supprime_Fils");
   end;
end T_Arbres_Binaires_D_Entiers;
```

Exécution (1)


```
$ t_arbres_binaires_d_entiers
Nombre d'elements de A      = 0
profondeur de A             = 0
L'element courant est vide = FALSE
Suis-je sur une feuille     = TRUE
Suis-je sur un noeud        = FALSE
L'element courant est vide = FALSE
Suis-je sur une feuille     = FALSE
Suis-je sur un noeud        = TRUE
Tentative d'ajout sans deplacement...
Levee de ARBRES_BINAIRES_D_ENTIERS.POSITION_NON_VIDE_ERROR dans
   Ajoute_Noeud
L'element courant est vide = FALSE
Suis-je sur une feuille     = TRUE
Suis-je sur un noeud        = FALSE
L'element courant est vide = FALSE
Suis-je sur une feuille     = FALSE
Suis-je sur un noeud        = TRUE
L'element courant est       = 8
Nombre d'elements de A      = 11
profondeur de A             = 6
```

Exécution (2)


```
profondeur Minimale de A    = 2
L'element courant est       = 1
Nombre d'elements de A      = 8
profondeur de A             = 5
profondeur Minimale de A    = 2
Nombre d'elements de B      = 8
profondeur de B             = 5
profondeur Minimale de B    = 2
A et B sont-ils egaux       = TRUE
A et B sont-ils egaux       = FALSE
A et B sont-ils egaux       = FALSE
A et B sont-ils egaux       = TRUE
Sur un arbre vide...
Levee de ARBRES_BINAIRES_D_ENTIERS.ELEMENT_COURANT_NULL_ERROR dans
   Etiquette_Element_Courant
Levee de ARBRES_BINAIRES_D_ENTIERS.ELEMENT_COURANT_NULL_ERROR dans
   Element_Courant_Est_Feuille
Levee de ARBRES_BINAIRES_D_ENTIERS.ELEMENT_COURANT_NULL_ERROR dans
   Element_Courant_Est_Noeud
Levee de ARBRES_BINAIRES_D_ENTIERS.ELEMENT_COURANT_NULL_ERROR dans
   Supprime_Fils
```

En guise de conclusion

- Un paquetage est réutilisable, ce qui implique des précautions
  - Se poser des questions sur les aspects «réutilisation»
    - Bonne encapsulation des types
    - Bonne définition des primitives de manipulation associées
    - Bon paramétrage des données manipulées
    - Bonne documentation (commentaires)

- Ces observations sont encore plus vraies pour une unité générique
  - On doit considérer en plus le paramétrage de l'unité
    - Hypothèses sur les paramètres «importés» par instanciation
    - Relations entre les types formels génériques
    - Relations entre les sous-programmes formels génériques et les types formels génériques