# 21 - Structures de données récursives

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

PrC

# Définition

- Structures de données récursives
  - La définition de la structure référence la structure elle-même
  - Extrêmement classique dans le cas de l'utilisation de pointeurs
    - Listes
    - Arbres
    - etc.
- Peut nécessiter une «déclaration à l'avance» des structures qui seront redéfinies ensuite
  - Par exemple pour définir le type «pointeur sur quelque chose», il faut savoir que le «quelque chose» va être défini
- En Ada, on a une situation similaire à celle des constantes différées
  - Déclaration d'un type sans description
  - Définition de ce type plus loin

# Notion de déclaration incomplète

Similaire à la notion de déclaration de constantes différées

```
type Cellule;  -- Déclaration du type
type Lien is access Cellule;  -- Définition du type pointeur
type Cellule is record -- Définition complète du type
   Valeur : Integer;
   Succ   : Lien := null;
   Pred   : Lien := null;
end record;
```

# Exemple de type «récursif croisé»



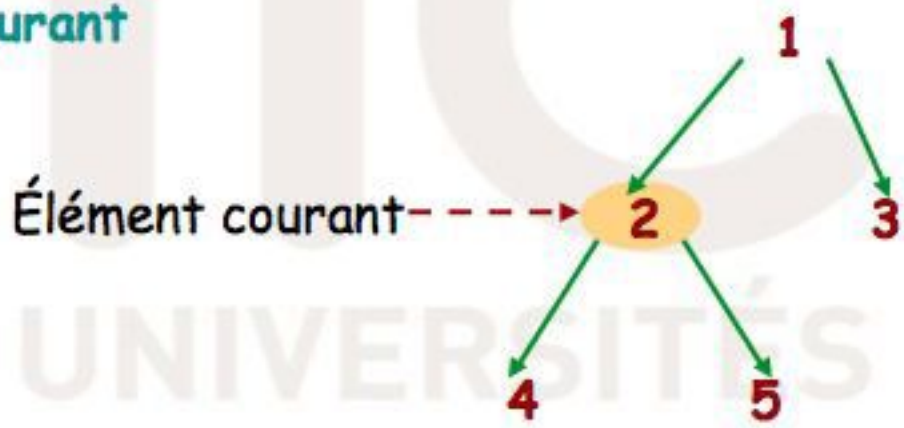
```
type Maison; -- Déclaration du type Maison
type Genre is (Homme Femme);
type Humain (Sexe : Genre); -- Déclaration du type Humain
-- Déclaration des pointeurs sur les types Maison et Humain
type Chez_soi is access Maison;
type Individu is access Humain;
-- Le type Maison
type Maison is record
   Adresse : String (1 .. 50);
   Proprio : Individu;
end record;
-- Le type Humain
type Humain (Sexe : Genre) is record
   Nom     : String (1 .. 20);
   Reside : Chez_Soi;
   case Sexe is
      when Homme => null;
      when Femme =>
         Epouse : String (1 .. 20);
   end case;
end record;
```

# Exemple: un gestionnaire d'arbres binaires d'entiers



## Objectifs

- Gestionnaire d'arbres binaires d'entiers
- Fonctions
  - Création/Destruction
  - Récupération d'informations (nombre d'éléments, profondeur, etc.)
  - Navigation dans l'arbre (notion d'élément courant)
  - Récupérer l'étiquette de l'élément courant
  - Ajouter des nœuds
  - Supprimer des sous-arbres
  - Recopie, comparaison d'égalité




## Choix

- Utilisation de pointeurs
- Définition à travers un type abstrait de données
  - On souhaite redéfinir la comparaison d'égalité et l'affectation
  - L'usage de pointeurs rend la sémantique par défaut de ces opérations dangereuse

# Spécification de Arbres_Binaires_D_Entiers (1)



```
-- Gestionnaire d'arbres binaires d'entiers
package Arbres_Binaires_D_Entiers is

   -- 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_D_Entiers is limited private;

   -- Créer un arbre binaire vide
   procedure Creer (A : out Un_Arbre_Binaire_D_Entiers);

   -- Détruire un arbre binaire
   procedure Detruire (A : in out Un_Arbre_Binaire_D_Entiers);

   -- Récupérer le nombre d'éléments d'un arbre binaire
   function Nombre_D_Elements (A : in Un_Arbre_Binaire_D_Entiers)
        return Natural;
```

arbres_binaires_d_entiers.ads

# Spécification de Arbres_Binaires_D_Entiers (2)



```
-- Récupérer la profondeur d'un arbre binaire
function Profondeur (A : in Un_Arbre_Binaire_D_Entiers)
      return Natural;
-- Récupérer la profondeur minimale d'un arbre binaire
function Profondeur_Min (A : in Un_Arbre_Binaire_D_Entiers)
      return Natural;
-- Se déplacer dans l'arbre
procedure Aller_Racine (A : in out Un_Arbre_Binaire_D_Entiers);
procedure Descendre (A : in out Un_Arbre_Binaire_D_Entiers;
                     D : in Direction);
-- Savoir où on se trouve
function Element_Courant_Est_Feuille
        (A : in Un_Arbre_Binaire_D_Entiers) return Boolean;
function Element_Courant_Est_Nœud
        (A : in Un_Arbre_Binaire_D_Entiers) return Boolean;
function Element_Courant_Est_Vide
        (A : in Un_Arbre_Binaire_D_Entiers) return Boolean;
-- Récupérer la valeur de l'élément courant
function Etiquette_Element_Courant
        (A : in Un_Arbre_Binaire_D_Entiers) return Integer;
```

# Spécification de Arbres_Binaires_D_Entiers (3)



```
-- Ajouter un élément à une direction donnée que l'on suppose vide
-- dans l'élément courant
procedure Ajoute_Noeud (V : in Integer;
                        D : in Direction;
                        A : in out
Un_Arbre_Binaire_D_Entiers);

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

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

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

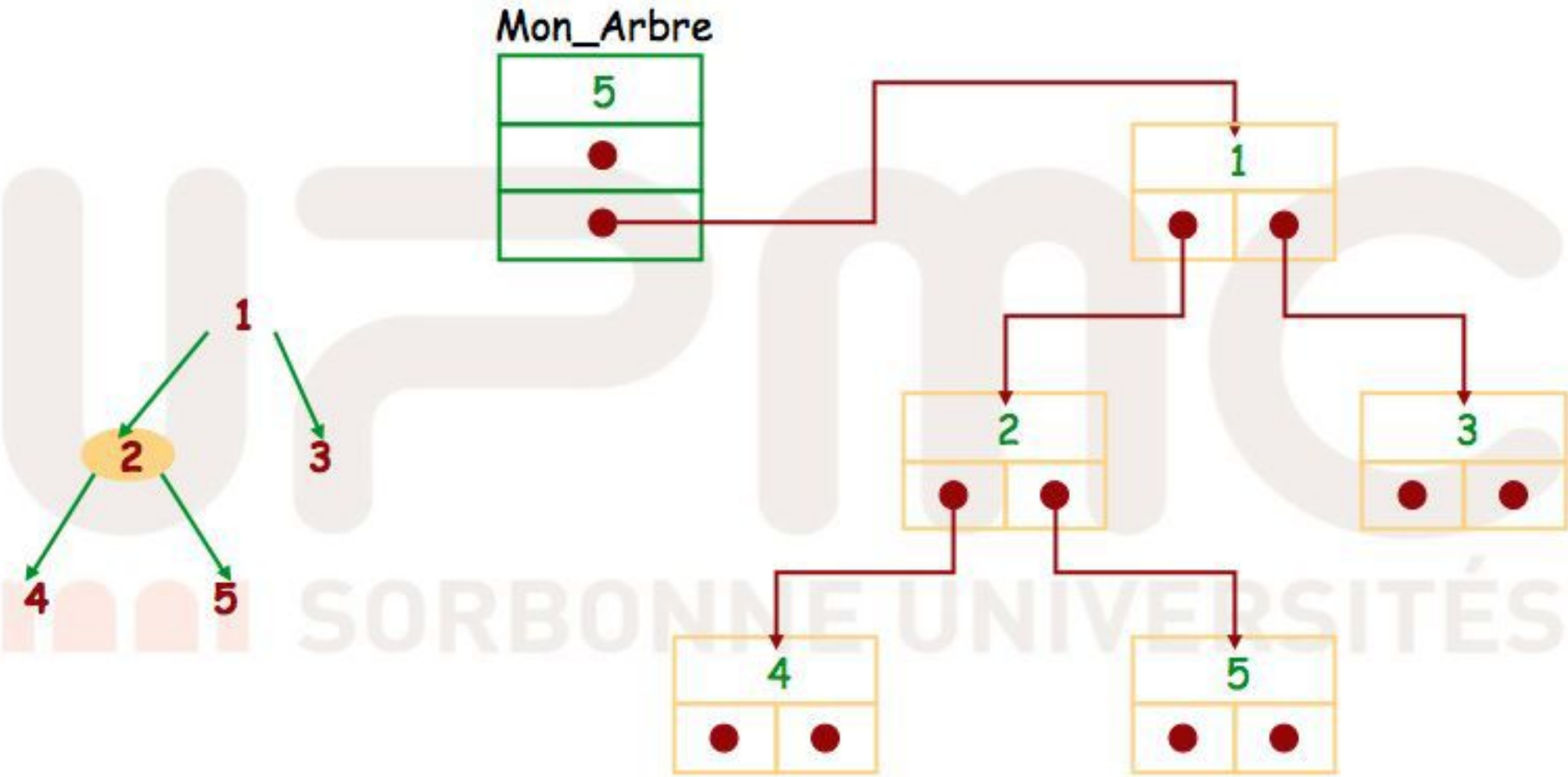
# Spécification de Arbres_Binaires_D_Entiers (4)



```
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 : Integer;
      Descendance : Tab_Fils := (null, null);
   end record;

   type Un_Arbre_Binaire_D_Entiers 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_D_Entiers;
```

# Visualisation de la structure Un_Arbre_Binaire_D_Entiers





# Corps de Arbres_Binaires_D_Entiers (1)



```
with Unchecked_deallocation;
with Ada.Text_Io; use Ada.Text_Io;
package body Arbres_Binaires_D_Entiers is
   -- Création de la procédure de désallocation (mécanisme vu plus tard)
   procedure Desalloue_Nœud
         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.all.Descendance (Gauche), N1);
         Desalloue_Recursivement (An.all.Descendance (Droite), N2);
         Desalloue_Noeud (An);
         Nb_El := N1 + N2 + 1;
      else
         Nb_El := 0;
      end if;
   end Desalloue_Recursivement;
```

arbres_binaires_d_entiers.adb

# Corps de Arbres_Binaires_D_Entiers (2)



```
procedure Creer (A : out Un_Arbre_Binaire_D_Entiers) is

begin
   A := (0, null, null);
end Creer;

procedure Detruire (A : in out Un_Arbre_Binaire_D_Entiers) is

   Nb_El : Natural;

begin
   Desalloue_Recursivement (A.Racine, Nb_El);
   A := (0, null, null);
end Detruire;

function Nombre_D_Elements
   (A : in Un_Arbre_Binaire_D_Entiers) return Natural is

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

# Corps de Arbres_Binaires_D_Entiers (3)



```
function Profondeur (A : in Un_Arbre_Binaire_D_Entiers)
   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.all.Descendance (Gauche)),
                          Calcule_Profondeur
                             (An.all.Descendance (Droite)))
                + 1;
   end Calcule_Profondeur;

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

# Corps de Arbres_Binaires_D_Entiers (4)



```
function Profondeur_Min (A : in Un_Arbre_Binaire_D_Entiers)
   return Natural is

   function Calcule_Profondeur (An : A_Noeud) return Natural is

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

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

# Corps de Arbres_Binaires_D_Entiers (5)



```
procedure Aller_Racine (A : in out Un_Arbre_Binaire_D_Entiers) is

begin
   A.Element_Courant := A.Racine;
end Aller_Racine;

procedure Descendre (A : in out Un_Arbre_Binaire_D_Entiers;
                     D : in Direction) is

begin
   A.Element_Courant := A.Element_Courant.all.Descendance(D);
end Descendre;

function Element_Courant_Est_Feuille
   (A : in Un_Arbre_Binaire_D_Entiers) return Boolean is

begin
   return A.Element_Courant.all.Descendance(Gauche) = null and
          A.Element_Courant.all.Descendance(Droite) = null;
end Element_Courant_Est_Feuille;
```

# Corps de Arbres_Binaires_D_Entiers (6)



```
function Element_Courant_Est_Noeud
   (A : in Un_Arbre_Binaire_D_Entiers) return Boolean is

begin
   return A.Element_Courant.all.Descendance(Gauche) /= null or
          A.Element_Courant.all.Descendance(Droite) /= null;
end Element_Courant_Est_Noeud;

function Element_Courant_Est_Vide
   (A : in Un_Arbre_Binaire_D_Entiers) return Boolean is

begin
   return A.Element_Courant = null;
end Element_Courant_Est_Vide;

function Etiquette_Element_Courant
   (A : in Un_Arbre_Binaire_D_Entiers) return Integer is

begin
   return A.Element_Courant.all.Contenu;
end Etiquette_Element_Courant;
```

# Corps de Arbres_Binaires_D_Entiers (7)



```
procedure Ajoute_Noeud (V : in Integer;
                        D : in Direction;
                        A : in out Un_Arbre_Binaire_D_Entiers) is
   N: Natural;
begin
    if A.Nombre_D_Elements = 0 then
       A.Racine := new Noeud'(Contenu => V,
                              Descendance => (others => null));
       A.Element_Courant := A.Racine;
    else
       -- supprime les elements existants
       Desalloue_Recursivement
              (A.Element_Courant.all.Descendance (D), N);
       A.Element_Courant.all.Descendance (D) :=
            new Noeud'(Contenu => V,
                       Descendance => (others => null));
    end if;
    A.Nombre_D_Elements := A.Nombre_D_Elements + 1 - N;
end Ajoute_Noeud;
```

# Corps de Arbres_Binaires_D_Entiers (8)



```
procedure Supprime_Fils (A : in out Un_Arbre_Binaire_D_Entiers;
                         D : in Direction) is

   Nb_El : Natural;

begin
    Desalloue_Recursivement
       (A.Element_Courant.all.Descendance (D), Nb_El);
    A.Nombre_D_Elements := A.Nombre_D_Elements - Nb_El;
end Supprime_Fils;
```

# Corps de Arbres_Binaires_D_Entiers (9)



```
procedure Copie (A1 : in Un_Arbre_Binaire_D_Entiers;
                 A2 : out Un_Arbre_Binaire_D_Entiers) is

   function Copie_Recursive (An : in A_Noeud) return A_Noeud is

   begin
      if An = null then
         return null;
      else
         return new Noeud'(An.Contenu,
                           (Copie_Recursive (An.all.Descendance (Gauche)),
                            Copie_Recursive (An.all.Descendance (Droite))));
      end if;
   end Copie_Recursive;

  Tmp : Natural;

begin -- Copie
   A2.Nombre_D_Elements := A1.Nombre_D_Elements;
   Desalloue_recursivement(A2.Racine, Tmp);-- Au cas ou l'arbre ne serait pas déjà vide!
   A2.Racine := Copie_Recursive (A1.Racine);
   A2.Element_Courant := A2.Racine;
end Copie;
```

# Corps de Arbres_Binaires_D_Entiers (10)



```
function "=" (A1, A2 : in Un_Arbre_Binaire_D_Entiers)
   return Boolean is

   function Comparaison_Recursive (An1, An2 : in 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.all.Descendance (Gauche)) and then
                Comparaison_Recursive (An1.all.Descendance (Droite),
                      An2.all.Descendance (Droite));
      end if;
   end Comparaison_Recursive;
```

# Corps de Arbres_Binaires_D_Entiers (11)



```
   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_D_Entiers;
```

# Test de Arbres_Binaires_D_Entiers (1)



t_arbres_binaires_d_entiers.adb

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

procedure T_Arbres_Binaires_D_Entiers is

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)));
   -- 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)));
```

# Test de Arbres_Binaires_D_Entiers (2)



```
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);
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);
Descendre (A, Gauche);
```

# Test de Arbres_Binaires_D_Entiers (3)



```
Ajoute_Noeud (10, Gauche, A);
Descendre (A, Gauche);
Ajoute_Noeud (11, Gauche, A);
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)));
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)));
```

# Test de Arbres_Binaires_D_Entiers (4)



```
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)));
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)));
```

# Test de Arbres_Binaires_D_Entiers (5)



```
   Put_Line ("profondeur Minimale de B   =" &
             Natural'Image (Profondeur_Min (B)));
   Put_Line ("A et B sont-ils egaux      = " &
             Boolean'Image (A = B));
   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));
end T_Arbres_Binaires_D_Entiers;
```

# Exemple d'exécution

```
$ 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
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
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
```