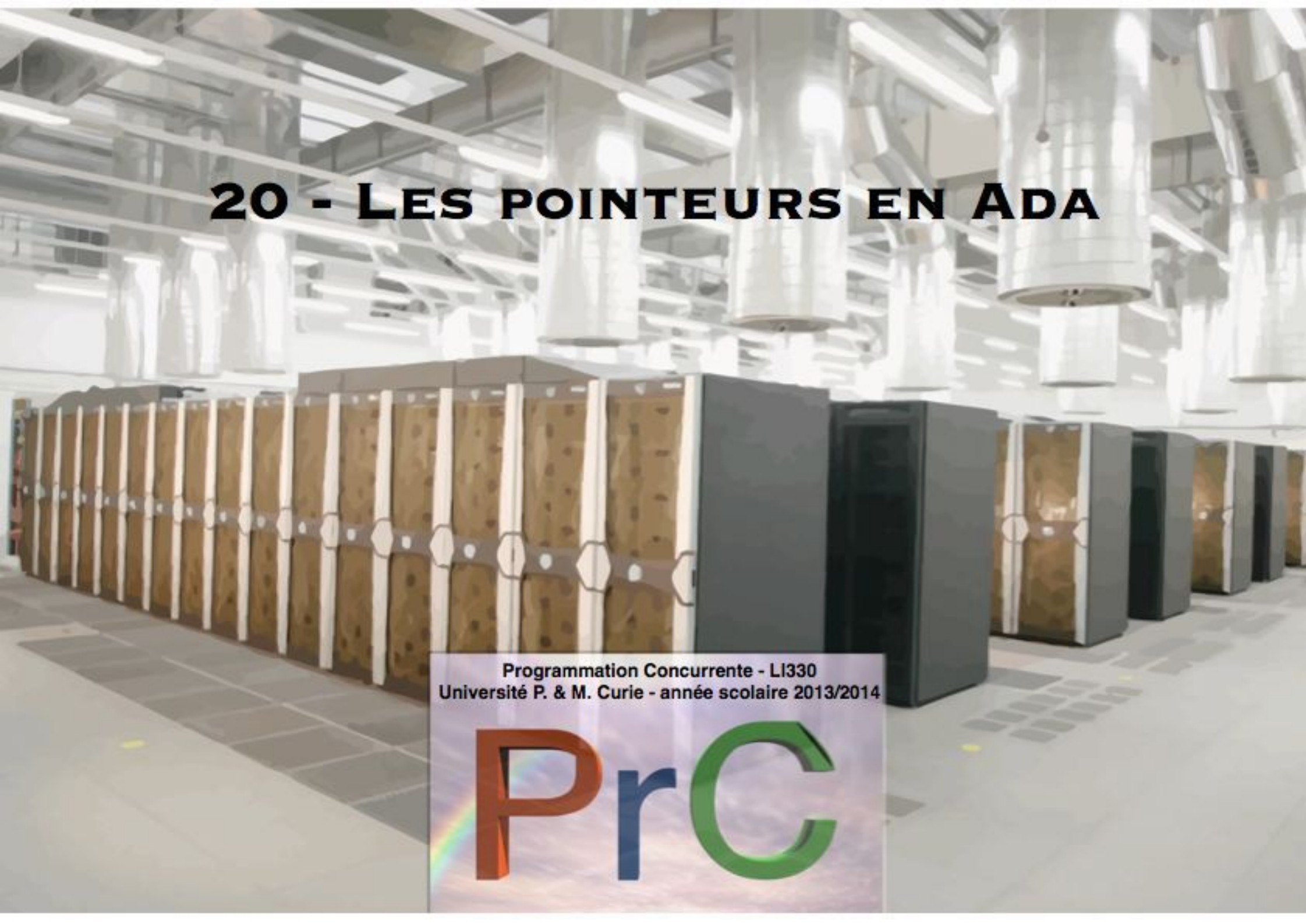
20 - Les pointeurs en Ada



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

PrC

Allocation/désallocation

- **Allocation**
  - Instruction new
    - On indique le sous-type de l'objet pointé
    - On peut affecter une valeur initiale (même pour les types composés)

- **Désallocation**
  - Construction d'un désallocateur pour chaque couple <type_pointeur, type_pointé>
  - Construction basée sur le principe de la généricité (sera vu plus tard)
    - Particularisation d'une procédure Unchecked_Deallocation
  - On transmet à la procédure particularisée l'accès à la zone mémoire à désallouer

Pointeurs en Ada

- **Pointeurs appelés «accès» en Ada**

  Décl_accès **type** identificateur **is access** id_type;

  Peut désigner un type non contraint

- **Les pointeurs en Ada sont typés**
  - En fonction du type de valeurs contenues dans l'espace mémoire correspondant
  - On n'étudiera pas la notion de «pointeur généralisé»
- **L'«élément neutre» est null**
- **Accès aux éléments pointés**

  Objet désigné complet (tableaux, articles, etc.)

  accès_cpl id_variable.**all**

  accès_comp id_variable.**[all.]**id_champ

Exemple: un gestionnaire de collection d'entiers

- Objectifs
  - Gestionnaire capable de contenir des «collections» d'entiers
    - Collection de taille maximale
    - Collection extensible sur demande
  - Opérations
    - Ajout d'éléments,
    - Recherche d'éléments
    - Informations sur la taille
    - Recopie
    - Comparaison
- 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 Collection_D_Entiers (1)


```
-- gestionnaire d'une collection d'entiers
package Collection_D_Entiers is

   -- le type collection (limité-privé)
   type Une_Collection_D_Entiers is limited private;

   Collection_D_Entiers_Vide : constant Une_Collection_D_Entiers;

   -- Créer une collection d'une taille maximum
   procedure Creer (T : in Positive;
                    C : out Une_Collection_D_Entiers);

   -- Agrandir une collection d'une taille maximum
   procedure Agrandir (T : in Positive;
                       C : in out Une_Collection_D_Entiers);
   -- Détruire une collection
   procedure Detruire (C : in out Une_Collection_D_Entiers);
```

collection_d_entiers.ads

Spécification de Collection_D_Entiers (2)


```
-- Récupérer la taille courante d'une collection
function Taille_Courante (C : in Une_Collection_D_Entiers)
         return Natural;
-- Récupérer la taille maximum d'une collection
function Taille_Max (C : in Une_Collection_D_Entiers)
         return Natural;
-- La collection est-elle pleine?
function Est_Pleine (C : in Une_Collection_D_Entiers)
         return Boolean;
-- Ajouter un élément dans une collection
procedure Ajoute_Element (E : in Integer;
                          C : in out Une_Collection_D_Entiers);
-- Retrouver la position d'un élément dans une collection
function Position_Element (E : in Integer;
                           C : in Une_Collection_D_Entiers)
         return Natural;
-- Supprimer un élément à une position donnée
procedure Supprime_Element (P : in Positive;
                            C : in out Une_Collection_D_Entiers);
```

Spécification de Collection_D_Entiers (3)


```
   -- Recopier une collection
   procedure Recopie (C1 : in Une_Collection_D_Entiers;
                      C2 : out Une_Collection_D_Entiers);

   -- Comparer deux collections
   function "=" (C1, C2 : in Une_Collection_D_Entiers)
            return Boolean;

private
   type Tableau is array (Positive range <>) of Integer;

   type A_Tableau is access Tableau;

   type Une_Collection_D_Entiers is record
      Taille_Courante : Natural := 0;
      Contenu : A_Tableau := null;
   end record;

   Collection_D_Entiers_Vide : constant Une_Collection_D_Entiers
                                     := (0, null);
end Collection_D_Entiers;
```

Corps de Collection_D_Entiers (1)

```
with Unchecked_deallocation;
package body Collection_D_Entiers is
   -- Création de la procédure de désallocation (mécanisme vu plus tard)
   procedure Desalloue_Tableau is new
             Unchecked_deallocation (Tableau, A_Tableau);

   procedure Creer (T : in Positive;
                    C : out Une_Collection_D_Entiers) is
   begin
      C := (0, new Tableau (1.. T));
   end Creer;

   procedure Agrandir (T : in Positive;
                       C : in out Une_Collection_D_Entiers) is


      T2 : A_Tableau := new Tableau (1 .. C.Contenu.all'Length + T);

   begin
      T2.all (1.. C.Contenu.all'Length) := C.Contenu.all;
      Desalloue_Tableau (C.Contenu);
      C.Contenu := T2;
   end Agrandir;
```

collection_d_entiers.adb

Corps de Collection_D_Entiers (2)

```
procedure Detruire (C : in out Une_Collection_D_Entiers) is
begin
   C.Taille_Courante := 0;
   Desalloue_Tableau (C.Contenu);
end Detruire;


function Taille_Courante (C : in Une_Collection_D_Entiers)
         return Natural is
begin
   return C.Taille_Courante;
end Taille_Courante;
```



```
function Taille_Max (C : in Une_Collection_D_Entiers)
         return Natural is
begin
   return C.Contenu.all'Length;
end Taille_Max;
```

Corps de Collection_D_Entiers (3)


```
function Est_Pleine (C : in Une_Collection_D_Entiers)
         return Boolean is
begin
   return C.Taille_Courante = C.Contenu.all'Length;
end Est_Pleine;



procedure Ajoute_Element (E : in Integer;
                          C : in out Une_Collection_D_Entiers) is
begin
   C.Contenu.all (C.Taille_Courante + 1) := E;
   C.Taille_Courante := C.Taille_Courante + 1;
end Ajoute_Element;
```

Corps de Collection_D_Entiers (4)


```
function Position_Element (E : in Integer;
                           C : in Une_Collection_D_Entiers)
         return Natural is
begin
   for I in C.Contenu.all'First .. C.Taille_Courante loop
      if C.Contenu.all (I) = E then
         return I;
      end if;
   end loop;
   return 0; -- 0 => pas trouvé
end Position_Element;


procedure Supprime_Element (P : in Positive;
                            C : in out Une_Collection_D_Entiers) is
begin
   -- On recopie la tranche supérieure du tableau (on pourrait juste permuter le Nième et
   -- me Pième élément
   C.Contenu.all (P ..C.Taille_Courante - 1) :=
            C.Contenu.all (P + 1 .. C.Taille_Courante);
   C.Taille_Courante := C.Taille_Courante - 1;
end Supprime_Element;
```

Corps de Collection_D_Entiers (5)


```
procedure Recopie (C1 : in Une_Collection_D_Entiers;
                   C2 : out Une_Collection_D_Entiers) is

begin
   Creer (C1.Contenu.all'length, C2);
   C2.Contenu.all := C1.Contenu.all;
   C2.Taille_Courante := C1.Taille_Courante;
end Recopie;
```

Corps de Collection_D_Entiers (6)


```
   function "=" (C1, C2 : in Une_Collection_D_Entiers)
            return Boolean is


   begin
      if C2.Taille_Courante /= C1.Taille_Courante then
         return false;
      elsif C1.Taille_Courante = 0 then
         return true; -- Pour protéger le .all
      else
         -- Si les tailles sont identiques, on doit comparer le contenu
         return   C1.Contenu.all (1..C1.Taille_Courante) =
                  C2.Contenu.all (1..C2.Taille_Courante);
      end if;
   end "=";

end Collection_D_Entiers;
```

Test de Collection_D_Entiers (1)


```
with Ada.Text_Io, Collection_D_Entiers;
use Ada.Text_Io, Collection_D_Entiers;

procedure Test_Collection_D_Entiers is

   A, B : Une_Collection_D_Entiers;

begin
   -- Création d'une première collection + test tailles
   Creer (10, A);
   Put_Line ("taille courante de A =" &
             Natural'Image (Taille_Courante (A)));
   Put_Line ("taille maximum de A  =" &
             Natural'Image (Taille_Max (A)));
   -- Agrandir et ajouter des éléments + test taille et = avec un élément vide
   Agrandir (5, A);
   Ajoute_Element (1, A);
   Ajoute_Element (2, A);
   Ajoute_Element (3, A);
   Ajoute_Element (4, A);
   Put_Line ("A est-il vide : " &
             Boolean'Image (A = Collection_D_Entiers.Vide));
```

test_collection_d_entiers.adb

Test de Collection_D_Entiers (2)


```
   Put_Line ("taille courante de A =" &
             Natural'Image (Taille_Courante (A)));
   Put_Line ("taille maximum de A  =" &
             Natural'Image (Taille_Max (A)));
   -- Recopie et test égalité sur 2 éléments semblables et non vides + test d'un élément
   -- présent
   Recopie (A, B);
   Put_Line ("2 est dans A a la position : " &
             Natural'Image (Position_Element (2, A)));
   Put_Line ("A et B sont egaux : " & Boolean'Image (A = B));
   -- Suppression d'un élément, test de l'égalité et de recherche d'un élément absent
   Supprime_Element (2, A);
   Put_Line ("2 est dans A a la position : " &
             Natural'Image (Position_Element (2, A)));
   Put_Line ("A et B sont egaux : " & Boolean'Image (A = B));
   -- Destruction et test de comparaison de deux collections vides
   Detruire (A);
   Put_Line ("A est-il vide : " &
             Boolean'Image (A = Collection_D_Entiers_Vide));
end Test_Collection_D_Entiers;
```

Exécution du test (pour information)

Exemple d'exécution

```
$ test_collection_d_entiers
taille courante de A = 0
taille maximum de A  = 10
A est-il vide : FALSE
taille courante de A = 4
taille maximum de A  = 15
2 est dans A a la position :  2
A et B sont egaux : TRUE
2 est dans A a la position :  0
A et B sont egaux : FALSE
A est-il vide : TRUE
```

Si "=" n'avait pas été surchargée

Comparaison d'égalité standard

- On compare le contenu de C1 et de C2
- Les deux zones mémoire sont différentes alors que leur contenu est identique
  - $@_1 \neq @_2$!

Si le type n'avait pas été limité(-privé)

Affectation standard

- On recopie C1 dans C2 sans recopier les espaces mémoire
- Conséquences:
  - L'espace désigné par $@_2$ est inaccessible
  - Toute modification ultérieure de la collection via C1 entraîne une modification (par effet de bord) sur le contenu de C2
  - Que se passera-t-il quand on détruira C1 ou C2?

Conseil sur les pointeurs

Attention

Il faut être extrêmement rigoureux dans la manipulation des pointeurs

① **Bien désallouer ce qui a été alloué (sauf si ramasse-miettes)**

② **Protéger si possible les utilisateurs d'unités basées sur des pointeurs en leur masquant l'implémentation sous-jacente**