Algorithmique & programmation

Chapitre 4 : Listes chaînées

Définitions

Pointeur

Cellule

Gestion de la mémoire

Mémoire statique & mémoire dynamique

- Les données d'un programme en cours d'exécution sont rangées dans 2 zones de mémoires distinctes :
  - la **mémoire statique** utilisée pour les données de taille constante connue du compilateur
  - la **mémoire dynamique** utilisée pour les données de taille susceptible de varier

Mémoire Statique

- Les langages de programmation sont tels que :
  - Les **variables globales** et les **variables locales** des sous-programmes ne peuvent contenir que des **données de taille constante**
  - Ces variables sont rangées dans la **mémoire statique**

- La **mémoire statique** est gérée comme une pile (stack)
  - à chaque appel d'un sous-programme, on réserve en sommet de pile la zone de mémoire nécessaire aux variables locales de ce sous-programme,
  - à chaque sortie de sous-programme, on libère cette mémoire

- Les variables locales à un sous programme n'occupent de la mémoire que pendant l'exécution de ce sous programme

Si l'on appelle successivement deux sous programmes, les variables locales de ces sous programmes occuperont les mêmes emplacements mémoires

Mémoire Statique : exemple

```
with Unchecked_Conversion;
with System,Ada.Text_Io, Ada.Integer_Text_Io;
use  System,Ada.Text_Io, Ada.Integer_Text_Io;

procedure Adresse is
  function Convertit is new
                          Unchecked_Conversion(Address,Integer);

  procedure Auxil is
    I:Integer;
  begin
    Put(Convertit(I'Address),Base=>16);
    New_Line;
  end Auxil;

begin
  Auxil;  -- écrit 16#100005C8#
  Auxil;  -- écrit 16#100005C8#
        -- i occupe les deux fois la même adresse
end Adresse;
```

Mémoire Dynamique

- La mémoire dynamique est occupée par des données
  - dont la taille peut varier d'une exécution du programme à l'autre et même au cours d'une même exécution
- Ces données sont toujours des valeurs manipulées à l'aide de **pointeurs**
- La mémoire dynamique est gérée comme un **tas** de mémoire (heap) dans lequel :
  - on puise de la mémoire (**allocation**) selon les besoins
  - on la remet à disposition (**libération**) lorsque l'on n'en a plus besoin

Mémoire dynamique

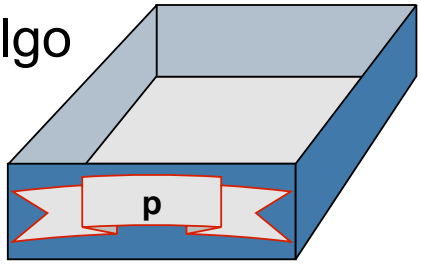
- On connaît les vecteurs
  - Une structure de données statique
    - Chaque vecteur occupe en mémoire la taille maximum envisagée
- Que peut-on faire si on ne souhaite pas perdre de place ?
  - Il faudrait une structure de donnée dynamique
    - À chaque instant, la place occupée par les données dépend uniquement de la taille de celles-ci
- On propose les listes linéaires chaînées

Le type pointeur

- Défintion

En ada	Dans un algorithme
`type Nom_Du_Type_Pointeur is access Nom_Du_Type_Des_Objets_Pointés;`	**type pointeur = ↑type ;**

- Exemple de déclaration en algo
  - Déclaration de type
    - **type point_t = ↑t ;**
  - Déclaration de variable
    - **p : point_t ;**



« boîte » vide de nom **p**
prête à recevoir une valeur
de type **point_t**

Quelle est la valeur d'une variable de type point_t ?

Le type pointeur

- Exemples de déclaration en ada

```
type Ta_Entier is access Integer;
                    --pointeur sur un entier
type Tr_Individu is record
  Nom:String(1..9);
  Age:Natural;
end record;
type Ta_Individu is access Tr_Individu;
      --pointeur sur un article Tr_Individu
type Tv_Ent is array(1..4) of Integer;
type Ta_Ent is access Tv_Ent;
           --pointeur sur un vecteur Tv_Ent
```

Le type pointeur

- Exemples de déclarations en algo
  - Déclaration d'un type "pointeur sur entier" :
    - **type tpoint_ent = ↑ entier;**
  - Déclaration d'un type "pointeur sur chaîne" :
    - **type tpoint_chaîne = ↑ chaîne;**
  - Déclaration d'un type "pointeur sur un type quelconque t" :
    - **type tpointeur = ↑ t;**

Valeur d'une variable de type pointeur

- Une variable de type pointeur contient l'adresse (un entier) de la cellule qui contient l'information
- Si
  - **type pointeur = ↑monType ;**
- et
  - **p : pointeur**
- alors
  - **p** contiendra un pointeur sur une cellule qui contient un élément de type **monType**

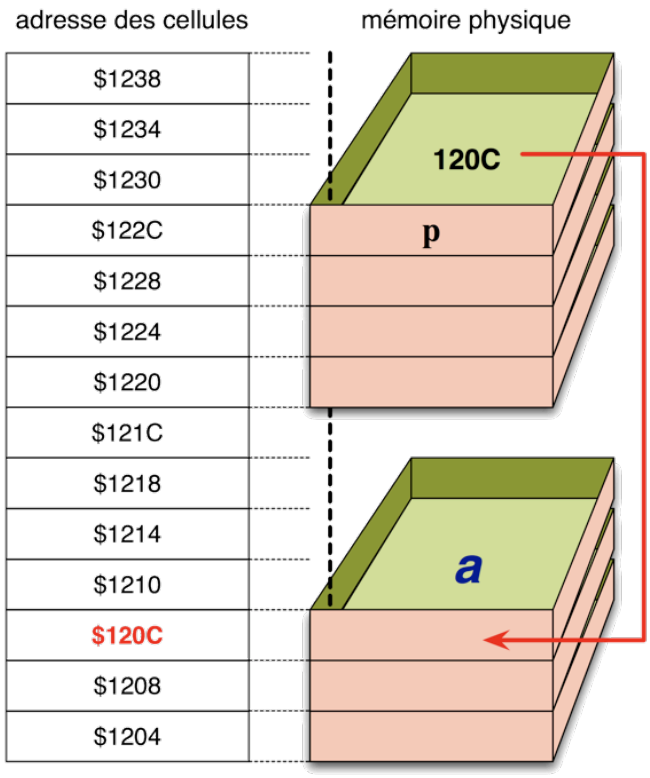
Valeur d'une variable de type pointeur

type pointeur = ↑monType;

- **monType** : type simple ou structuré

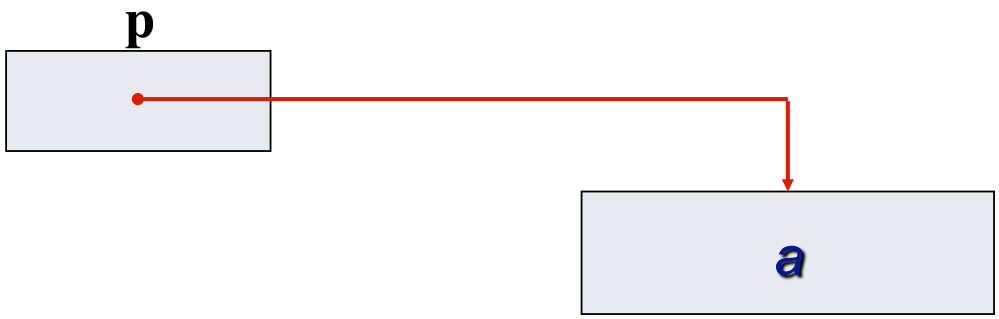
- **p** : variable de type **pointeur**, contient l'adresse de la cellule qui contient l'information ***a***

- **ici**
  - **p = $120C**
  - **p↑ = a**



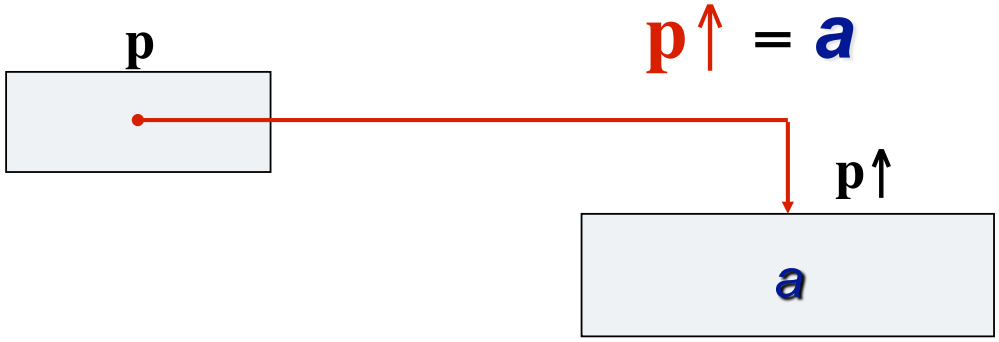
Valeur d'une variable de type pointeur

- Simplification graphique



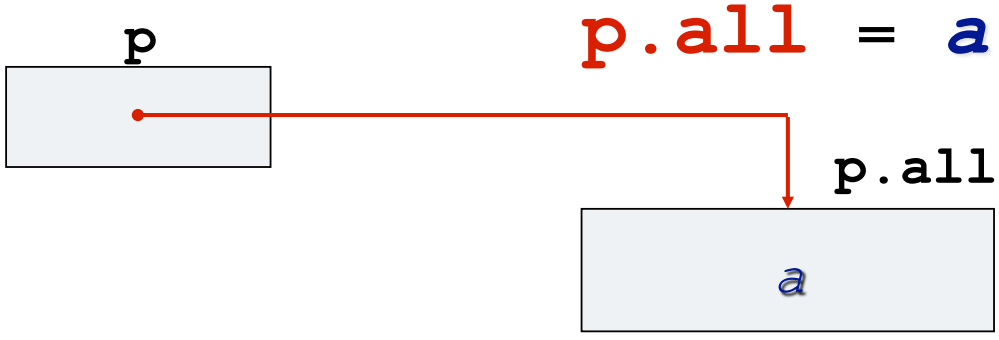
Accès au contenu d'une cellule (**algo**)

- **type pointeur = ↑monType ;**
  - **monType** : type simple ou structuré.
- **p** : variable de type **pointeur**, contient l'adresse de la cellule qui contient l'information ***a***
- **p↑** : désigne le contenu de la cellule pointée par **p**

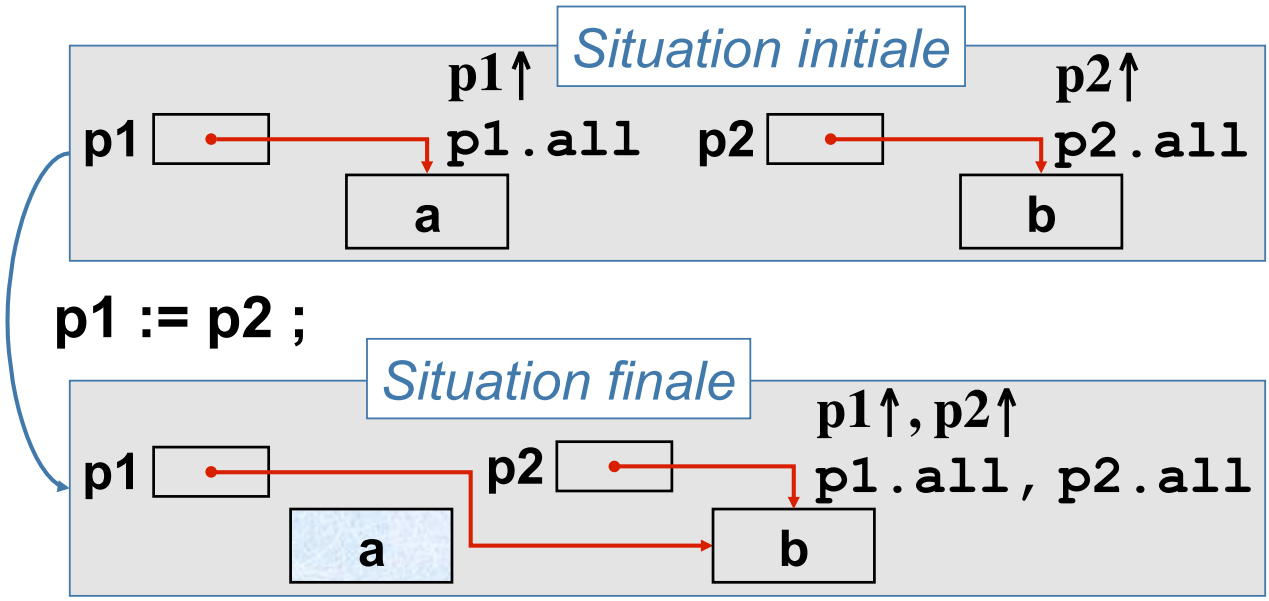


Accès au contenu d'une cellule (**ada**)

- `type Ta_pointeur is access monType ;`
  - `monType` : type simple ou structuré.
- `p` : variable de type `Ta_pointeur`, contient l'adresse de la cellule qui contient l'information ***a***
- `p.all` : désigne le contenu de la cellule pointée par `p`

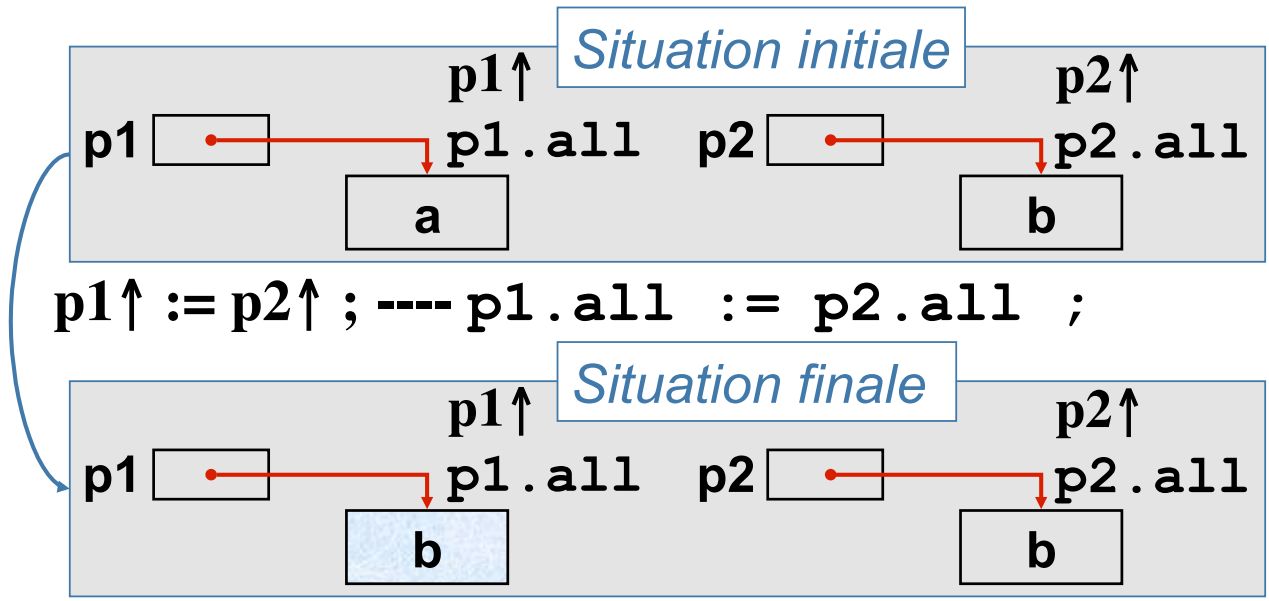


Attention au type manipulé !!



- **a** n'est plus accessible
- on peut atteindre **b** par **p1** ou **p2** (même adresse)

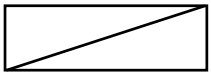
Attention au type manipulé !!



- **a** n'existe plus
- **b** existe en deux exemplaires avec **p1** et **p2**

Valeur d'adresse : **nil** ou **null**

- Une variable de type pointeur qui ne pointe sur aucune cellule est positionnée à la valeur particulière **nil** (algo) ou **null** (ada)

- **p := nil ; ---- p := null ;**
  - représentation graphique : **p** 

- **p↑** : n'est défini que si **p ≠ nil**
- **p.all** : n'est défini que si **p /= null**

Opérations sur les pointeurs en **ada** (1/2)

- Pour un **pointeur sur un article** on accède aux champs de l'article pointé par :
  - **nom_du_pointeur.all.nom_du_champ**

  ou
  - **nom_du_pointeur.nom_du_champ**

- Exemples
  - `Pind.all.Nom(1..4):="Toto" ;`
  - `Pind.Nom(3)='t' ;`
  - `Pind.all.Age:=12 ;`
  - `Pind.Age:=25 ;`

Opérations sur les pointeurs en **algo**

- Pour un **pointeur sur un article** on accède aux champs de l'article pointé par :
  - **nom_du_pointeur↑.nom_du_champ**

- Exemples
  - p↑.nom:="dupont" ;
  - p↑.age:=25 ;
  - p↑.moyenne=12 ;

Opérations sur les pointeurs en **ada** (2/2)

- Pour un **pointeur sur un tableau** on accède aux éléments du tableau pointé par :
  - **nom_du_pointeur.all(index)**

  ou
  - **nom_du_pointeur(index)**
- Remarque
  - On peut utiliser sur le tableau pointé les attributs usuels : `Length`, `First`, `Last`, `Range`
- Exemples
  - `Pvect(2)=14`
  - `Pvect'Length=4`

Gestion dynamique de la mémoire

On dispose d'un réservoir de cellules (tas)

- Besoin de mémoriser une information
  - demander une cellule en algo

procédure nouveau (r p : pointeur) ;
spécification *{tas non vide}* ➔ *{p contient l'adresse d'une nouvelle cellule allouée}*

  - demander une cellule en ada

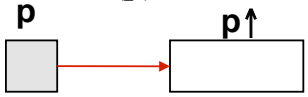
opérateur **new**

Utilisation de nouveau

- Déclaration d'un type pointeur sur entier :
  - type tpoint_ent = ↑ entier ;
- Déclaration d'une variable de ce type :
  - p : = tpoint_ent ;

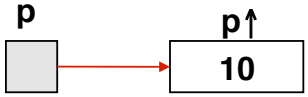
L'espace nécessaire au stockage de p est réservé
p n'a encore aucune valeur

- Affectation d'une valeur à p (adresse en mémoire dynamique d'une variable de type entier) :
  - Nouveau(p) ;



Un espace est réservé dans le tas pour stocker un entier :
p a pour valeur l'adresse de cet espace
L'espace lui-même est nommé p↑

- Affectation d'une valeur à p :
  - p↑ := 10;



p↑ prend pour valeur 10

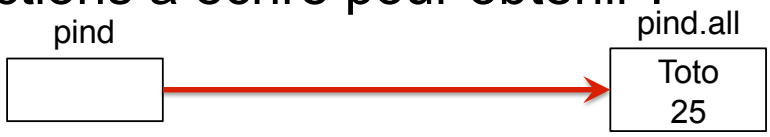
Utilisation de **new**

```
Nom_De_Pointeur:= new Nom_Du_Type_Des_Objets_Pointés;
```

```
type Ta_Entier is access Integer;

Pn : Ta_Entier := null ;
                              -- Pn a pour valeur null

Pn := new Integer;
        -- valeur de Pn : une adresse de contenu indéfini

Pn.all := 10;
 -- 10 est rangé à l'adresse contenue dans la variable Pn
```


Pointeur sur un article

- Instructions à écrire pour obtenir :




```
type Tr_Personne is record
   Nom:String(1..9);
   Age:Natural;
end record;
type Ta_Personne is access Tr_Personne;

Pind:Ta_Personne;
…
Pind:=new Tr_Personne;
Pind.Nom:=(1..4=>"Toto", others=>' ');
Pind.Age:=25;
```

Pointeur sur un tableau

- Instructions à écrire pour obtenir :


```
type Tv_Ent is array(1..4) of Integer;
type Ta_Ent is access Tv_Ent;

Pvect : Ta_Ent;
Pvect:=new Tv_Ent;
Pvect(1..4):=(41,14,32,23);
```


Pointeurs sur des tableaux non contraints

- Dans le cas de types pointeurs sur des tableaux non contraints, il est nécessaire de préciser, lors de la création de l'objet tableau pointé, quelle place maximale il va occuper en mémoire dynamique :

```
Pointeur:=new Type_Tableau_Non_Contraint(Intervalle);
```

- Exemple :

```
type Tv_Ent is array(Positive range <>) of Integer;
type Ta_Ent is access Tv_Ent;
...
get(N);
Pv:=new Tv_Ent(1..N); -- alloue un tableau de N entiers
```

- Les pointeurs sur des tableaux non contraints permettent de créer des objets dont la taille maximale est inconnue lors de la compilation.
- Il est possible d'adapter la taille d'un tableau si celle-ci devient inadaptée au cours de l'exécution du programme.

Allocation avec initialisation

- On peut aussi donner une valeur initiale lors de la création de l'objet pointé :

```
Nom_De_Pointeur :=
                new Nom_Du_Type_Des_Objets_Pointés'(Valeur);
```

- Exemples

```
Pn    := new Integer'(10);
Pind  := new Tr_Individu'((Nom=>"Toto      ",
                           Age=>25,
                           null));
Pvect:= new Tv_Ent'(41,14,32,23);
```


Mémoire dynamique pointée par une variable locale

- Soit le programme suivant :

```
with Unchecked_Conversion;
with Ada.Text_Io,Ada.Integer_Text_Io;
use  Ada.Text_Io,Ada.Integer_Text_Io;

procedure Adresse is
  type Ta_Entier is access Integer;
  function Convertit is new
                   Unchecked_Conversion(Ta_Entier,Integer);
  procedure Auxil is
    P:Ta_Entier:=new Integer;
  begin
    Put(Convertit(P),Base=>16); New_Line;
  end Auxil;
begin
  Auxil;   -- écrit 16#3FF80#
  Auxil;   -- écrit 16#3FF90#
  Auxil;   -- écrit 16#3FFA0#  p a des valeurs différentes !!
end Adresse;
```

Mémoire dynamique pointée par une variable locale

- La mémoire allouée par **new** n'est pas prise dans la pile mais dans le tas de mémoire dynamique
- cette mémoire n'est **pas libérée automatiquement**
  - Si l'on appelle 500 fois la procédure **Auxil** ces fragments de mémoire ne seront jamais libérés et la mémoire dynamique deviendra très engorgée
- Pour résoudre ce problème **on doit libérer la mémoire** pointée par **P** en fin de sous programme

Mémoire dynamique pointée par une variable locale

- Cette libération peut être effectuée automatiquement par un programme appelé ramasse miettes (garbage collector), ou être accomplie par les soins du programmeur.
- En **ada**, pour libérer soi-même un espace précédemment alloué par **new** on doit **instancier la procédure générique**

```
unchecked_deallocation(type_de_l'objet_pointé,
                                        type_du_pointeur)
```

- On utilise ensuite la procédure ainsi créée pour libérer la mémoire pointée par le pointeur
- La taille de nos TP est trop modeste pour nous obliger à cette gymnastique et nous ne ferons que rarement de libération en Ada
- Soyez conscients du problème, avec certains langages (comme C++) on remplit très facilement la mémoire dynamique sans même s'en rendre compte ...

Exemple

```
with Unchecked_Conversion,Unchecked_Deallocation;
with Ada.Text_Io,Ada.Integer_Text_Io;
use  Ada.Text_Io,Ada.Integer_Text_Io;

procedure Adresse is
   type Ta_Entier is access Integer;
   function Convertit is new
                          Unchecked_Conversion(Ta_Entier,Integer);
   procedure Liberer is new
                         Unchecked_Deallocation(Integer,Ta_Entier);
   procedure Auxil is
     P:Ta_Entier:=new Integer;
   begin
     Put(Convertit(P),Base=>16); New_Line;
     Liberer(P);
   end Auxil;
begin
   Auxil; -- écrit 16#3FFC8#
   Auxil; -- écrit 16#3FFC8#  P a les mêmes valeurs !!
end Adresse;
```

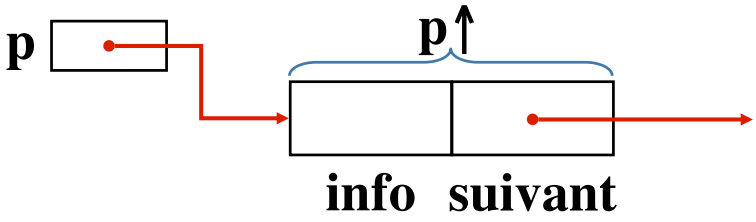

Gestion dynamique de la mémoire

- Information contenue dans une cellule devenue inutile
  - rendre la cellule au tas

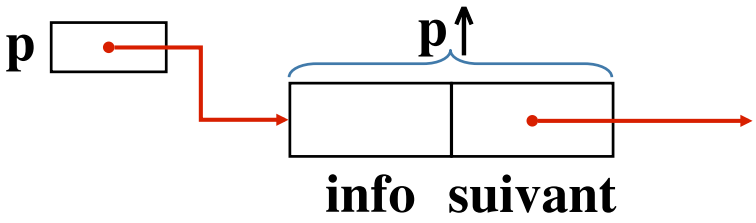
procédure laisser (d p : pointeur) ;
spécification { } ➔ *{rend la cellule d'adresse p au tas}*

Type cellule (élément de base d'une liste)

- **type cellule = structure**
  info : t ;
  suivant : pointeur ;
  fin ;
- **type pointeur = ↑cellule ;**
- avec type **t** : type simple, vecteur, structure, ...
- Une variable **p** de type pointeur contient l'adresse d'une cellule



Type cellule (élément de base d'une liste)



- Si **p ≠ nil** alors
  - la cellule pointée par **p** (cellule d'adresse **p**) est formée de deux champs :
    - **p↑.info** contient une information
    - **p↑.suivant** contient une adresse de cellule

Type cellule (élément de base d'une liste)

- En **ada**, on est obligé de faire une déclaration anticipée du type article

```
type Tr_Individu;
   -- déclaration anticipée de l'article

type Ta_Individu is access Tr_Individu;

type Tr_Individu is record
            Nom     : String(1..9);
            Age     : Natural;
            suivant: Ta_Individu;
end record;
```


Gestion dynamique & statique

- Vecteurs
  - gestion statique de la mémoire
    - lors de la création d'une variable de type vecteur, la place nécessaire au stockage d'un vecteur de longueur maximale est réservée en mémoire
- Pointeurs
  - gestion dynamique de la mémoire
    - le **programmeur** demande les cellules nécessaires à stocker des données
    - le **programmeur** libère les cellules lorsque les données stockées ne sont plus nécessaires