IF5

Algorithmique et Structures de Données

Akkouche Samir :
E-mail : samir.akkouche@liris.cnrs.fr
http://perso.univ-lyon1.fr/samir.akkouche/LIF5/

Knibbe Carole:
E-mail : carole.knibbe@bat710.univ-lyon1.fr

Le type abstrait LISTE D'ELEMENTS

Définition d'une liste

Liste = <> -- VIDE

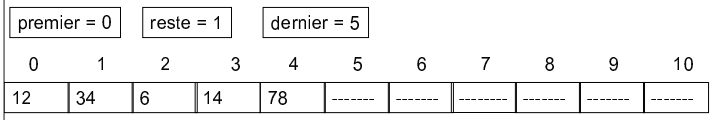
Liste = < Premier(*Elément*) | Reste(*Liste*) >

Structure de données à accès séquentiel

Opérations sur les Listes

- InitialiserListe //La liste est vide
- Listevide // teste si la liste est vide (renvoie vrai après l'initialisation)
- PremierElement // renvoie la valeur du premier élément de la liste
- AjouterenTete // Le nouvel élément devient le premier élément de la liste.
- SupprimerElement //Supprime un element

Stockage dans un tableau simple



Opérations sur les Listes:

- Initialiser : par exemple dernier = 0 ou premier = -1
- Listevide : renvoie vrai si dernier = 0
- PremierElement : renvoie la valeur en position 0
- AjouterenTete : Tous les éléments sont décalés à droite d'une position, dernier = dernier +1 , le nouvel élément est mis en 0.
- SupprimerElement : Repérer l'élément et décaler à gauche à partir de la position suivante et dernier = dernier -1

Stockage dans un tableau : autre solution

```
Cellule: structure
        info : Element
        suivant : entier
    fin structure
```

```
Liste: structure
        premier : entier
        place_libre : entier
        t : tableau [1..10000] de Cellule
    fin structure
```

Exemple de liste l : Liste

l.premier = 0

l.place_libre = 1

0	1	2	3	4	5	6	7	8	9	10
12	-------	-------	14	78	-------	-------	6	-------	-------	-------
3	2	5	7	-1	6	8	4	9	10	-1

Initialisation de la liste

l.premier = -1 l.place_libre = 0

0	1	2	3	4	5	6	7	8	9	10
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------
1	2	3	4	5	6	7	8	9	10	-1

Listevide : renvoie vrai si l.premier = -1

AjouterenTete : dans ce cas particulier,
l.t[l.place_libre].info ← valeur (23)
l.t[l.place_libre].suivant ← -1 (l.premier)
l.premier ← 0
l.place_libre ← 1

l.premier = 0 l.place_libre = 1

0	1	2	3	4	5	6	7	8	9	10
23	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------
-1	2	3	4	5	6	7	8	9	10	-1

Ajout en tête de la liste

l.premier = 0 l.place_libre = 1

0	1	2	3	4	5	6	7	8	9	10
12	-------	-------	14	78	-------	-------	6	-------	-------	-------
3	2	5	7	-1	6	8	4	9	10	-1

l.t[l.place_libre].info ← valeur (25),
temp ← l.premier (0)
l.premier ← l.place_libre (1)
l.place_libre ← l.t[l.place_libre].suivant (2)
l.t[l.premier].suivant ← temp (0)

l.premier = 1 l.place_libre = 2

0	1	2	3	4	5	6	7	8	9	10
12	25	-------	14	78	-------	-------	6	-------	-------	-------
3	0	5	7	-1	6	8	4	9	10	-1

Ajout en tête de la liste (suite)

l.premier = 1 l.place_libre = 2

0	1	2	3	4	5	6	7	8	9	10
12	25	-------	14	78	-------	-------	6	-------	-------	-------
3	0	5	7	-1	6	8	4	9	10	-1

l.t[l.place_libre].info ← valeur (44),
temp ← l.premier (1)
l.premier ← l.place_libre (2)
l.place_libre ← l.t[l.place_libre].suivant (5)
l.t[l.premier].suivant ← temp (1)

l.premier = 2 l.place_libre = 5

0	1	2	3	4	5	6	7	8	9	10
12	25	44	14	78	-------	-------	6	-------	-------	-------
3	0	1	7	-1	6	8	4	9	10	-1

Suppression d'un élément de la liste

l.premier = 2 | l.place_libre = 5

0	1	2	3	4	5	6	7	8	9	10
12	25	44	14	78	-------	-------	6	-------	-------	-------
3	0	1	7	-1	6	8	4	9	10	-1

Il faut chercher l'élément à supprimer (14) : id_elem ← 3
Il faut aussi connaitre son prédécesseur : id_pred ←0
l.t[id_pred].suivant ← l.t[id_elem] .suivant (7)
l.t[id_elem] .suivant ← l.t[l.place_libre]
l.place_libre = ind_elem (3)

l.premier = 2 | l.place_libre = 3

0	1	2	3	4	5	6	7	8	9	10
12	25	44	14	78	-------	-------	6	-------	-------	-------
7	0	1	5	-1	6	8	4	9	10	-1

Opérations sur les Listes (suite)

- AjouterenQueue// Le nouvel élément est en fin de liste
- RechercherunElement// renvoie « l'emplacement » éventuel de l'élément
- AfficherListe // Parcourt la liste et affiche les éléments
- …

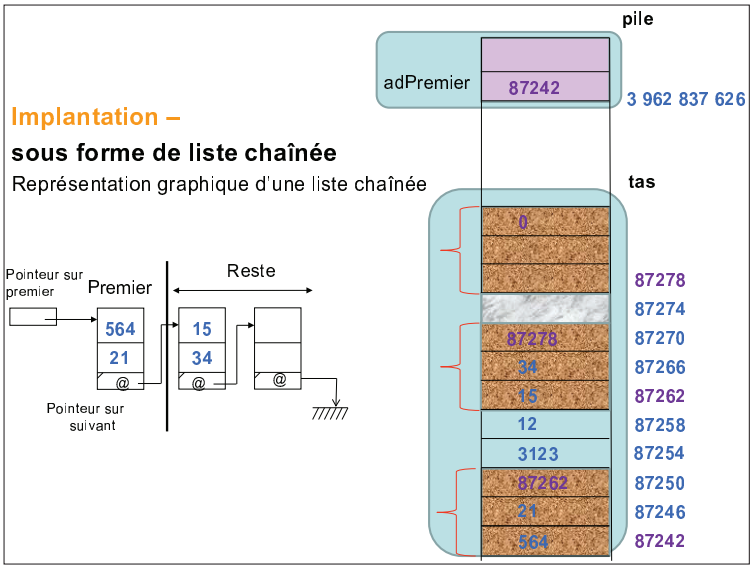
Autre méthode

LES LISTES CHAINEES

Implantation –

sous forme de liste chaînée

Représentation graphique d'une liste chaînée



Implantation des Listes chaînées (suite)
Fiche technique

```
Type
Cellule = stucture
            info : Element
            suivant : pointeur sur Cellule
          Fin structure
Liste = structure
            adPremiere: pointeur sur Cellule
        Fin structure
```

```
struct sCellule
        {
          Element  info ;
          struct  sCellule * suivant ;
        } ;
typedef struct sCellule Cellule;
struct sListe
    {
        Cellule * adPremiere;
    };
typedef struct sListe Liste;
```

TDA Liste

module Liste

importer **module** Element
exporter type **Cellule, Liste**
procedure initialiser(l : Liste)
Précondition : l n'est pas initialisée
Postcondition : l est une liste vide
Paramètre en mode donnée-résultat : l
procedure testament(l : Liste)
Précondition : l est bien initialisée
Postcondition : l est une liste vide
Paramètre en mode donnée résultat : l
----------A suivre----------
implantation

Finmodule

TDA Liste(suite)

module Liste(suite)

fonction estVide(l : Liste) : *Booleen*
Précondition : l est bien initialisée
Résultat : renvoie vrai si la liste est vide
Paramètre en mode donnée : l

fonction premierElement(l : Liste) : *Elemen*t
Précondition : l est bien initialisée, non vide
Résultat : renvoie l'élément situé en tête de liste
Paramètre en mode donnée : l
----------A suivre----------
implantation

Finmodule

TDA Liste (suite)

procedure ajouteEnTete(l : Liste, x : Element)
Précondition : l est bien initialisée
Postcondition : l'élément x est mis en tête de liste
Paramètre en mode donnée résultat : l
Paramètre en mode donnée : x

procedure ajouterEnQueue(l : Liste, x : Element)
Précondition : l est bien initialisée
Postcondition : l'élément x est mis en fin de liste
Paramètre en mode donnée résultat : l
Paramètre en mode donnée : x
----------A suivre----------

implantation

TDA Liste (suite)

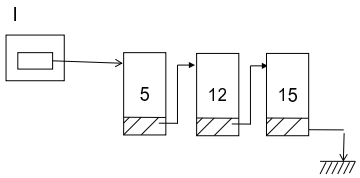
procedure afficher(l : Liste)
Précondition : l est bien initialisée
Postcondition : rien
Paramètre en mode donnée : l

fonction recherche(l : Liste, x : Element) : *pointeur sur Cellule*
Précondition : l est bien initialisée
Résultat : renvoie l'adresse de la cellule contenant x et NULL si x n'existe pas
Paramètres en mode donnée : l et x

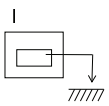
implantation

Implantation des Listes chaînées (suite)

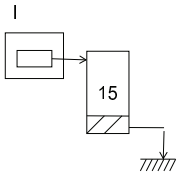
Exemple de manipulation de liste chaînée
on veut créer une liste d'entiers (arrivés dans l'ordre suivant : 15,12,5)



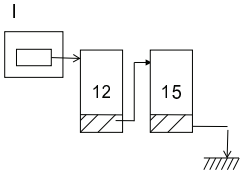
Etape 1: initialisation



Etape 2



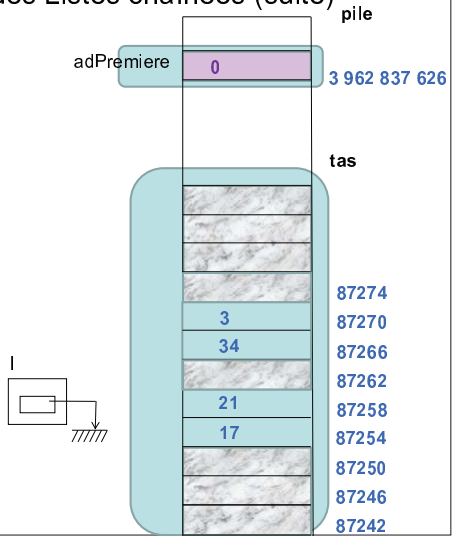
Etape 3



Implantation des Listes chaînées (suite)

```
procedure initialiser(l : Liste)
Donnée-résultat l
début
   l.adPremiere ← NULL;
fin initialiser
----------
a: Liste
initialiser (a)
```

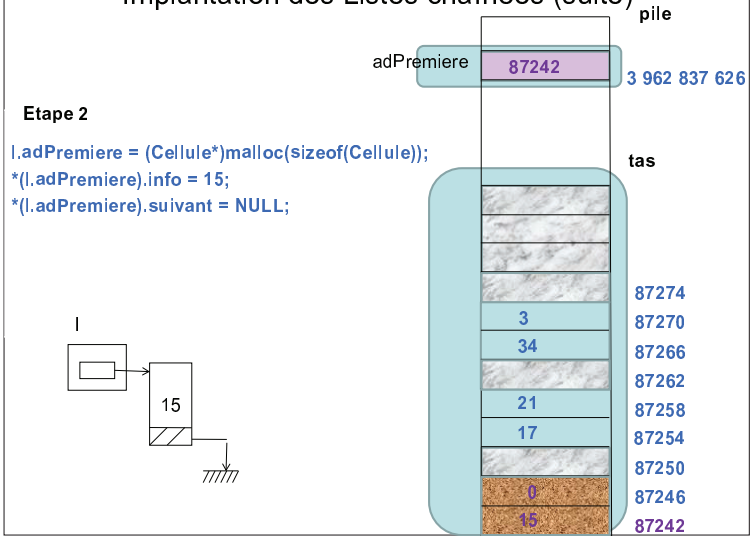
```
void initialiser(Liste *l)
{
   (*l).adPremiere = NULL;
}
----------
Liste a;
initialiser (&a);
```



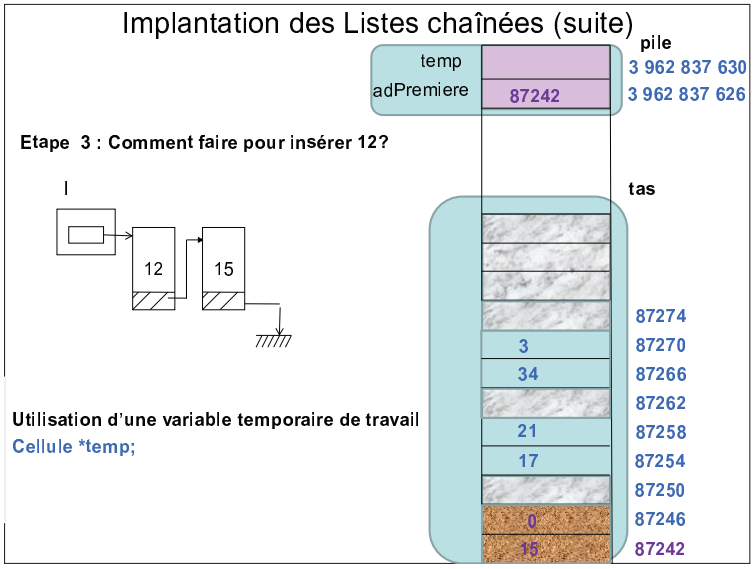
Implantation des Listes chaînées (suite)

Etape 2

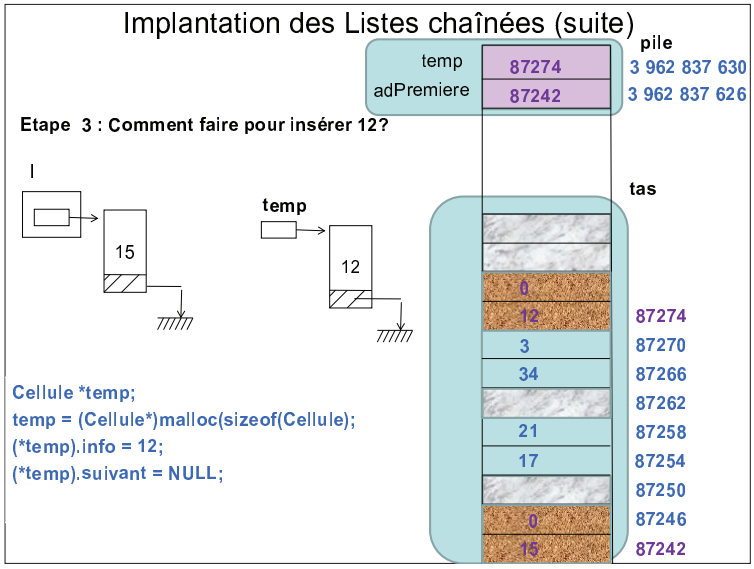
```
l.adPremiere = (Cellule*)malloc(sizeof(Cellule));
*(l.adPremiere).info = 15;
*(l.adPremiere).suivant = NULL;
```



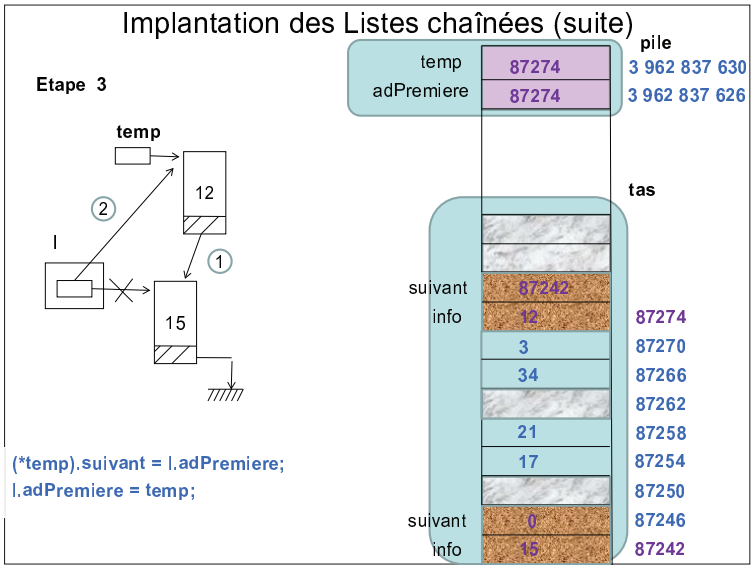
Implantation des Listes chaînées (suite)



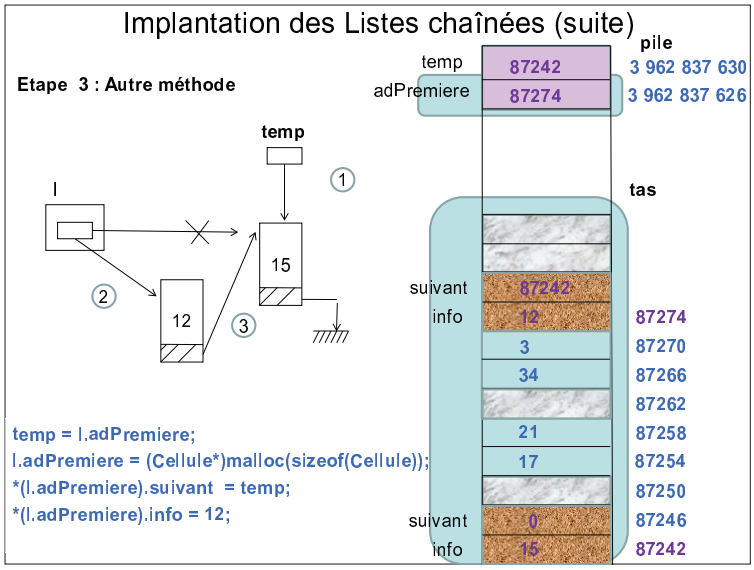
Implantation des Listes chaînées (suite)



Implantation des Listes chaînées (suite)



Implantation des Listes chaînées (suite)



Fiche technique
Initialisation

```
procedure initialiser(l : Liste)
Donnée-résultat l
début
   l.adPremiere = NULL;
fin initialiser
----------------------------------------
void initialiser(Liste *l)
{
   (*l).adPremiere = NULL;
}
```

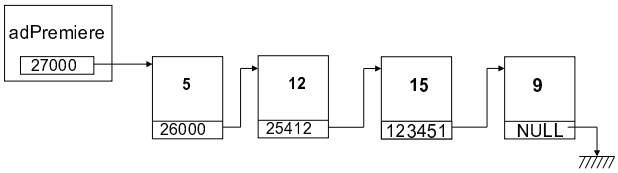
Fiche technique
Le chaînage en tête

Trois étapes

- Sauver l'adresse de la première cellule
- Réserver une cellule dans le tas avec le pointeur de la liste
- Chainer la nouvelle cellule à l'ancienne liste:

```
temp = l.adPremiere;
l.adPremiere = (Cellule*)malloc(sizeof(Cellule));
*(l.adPremiere).suivant  = temp;
*(l.adPremiere).info = 12;
```

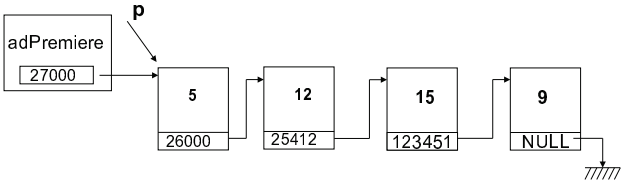
Libération d'une liste



- Problème : Comment libérer proprement toutes ces boites?

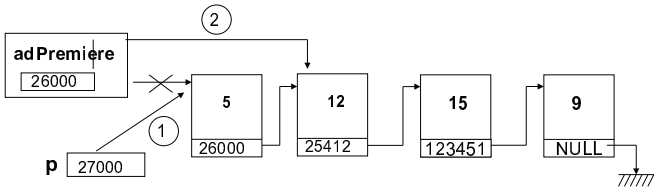
Libération d'une liste

- Méthode :
  1. On a besoin d'un pointeur de travail P
  2. Décrocher avec P la Cellule de tête sans perdre la liste
  3. Libérer la Cellule

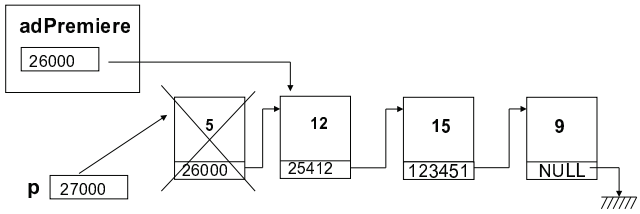


Libération d'une liste

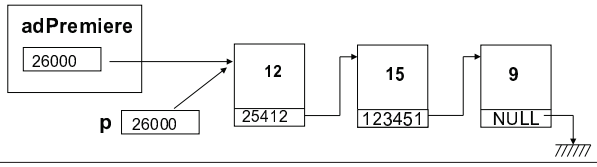
- Décrocher avec P la boite de tête sans perdre la liste
  - P ← l.adPremiere ①
  - l.adPremiere ← (l.adPremiere ↑.suivant) ②



Libération d'une liste



- Libérer la boite pointée par P :
  libere p → free (p);
- P ← l.adPremiere



Libération d'une liste

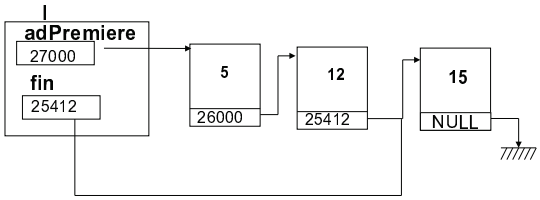
Algorithme :
Tant que l.adPremiere <> NULL faire
 p ← l.adPremiere
 l.adPremiere ← l.adPremiere ↑.suivant
 libere p
fin tantque

Traduction en C :

```
while(l->adPremiere !=NULL)
   {
      p = l->adPremiere;
      l->adPremiere = (l-> adPremiere)→suivant;
     free(p);
   }
```

Autre méthode de création d'une liste

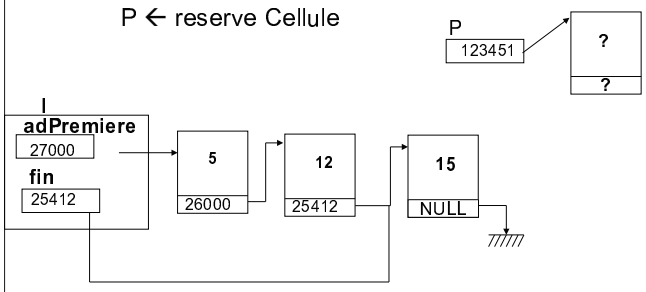
- Chaînage en queue : utilisation de trois pointeurs
  - adPremiere
  - fin
  - p



Autre méthode de création d'une liste

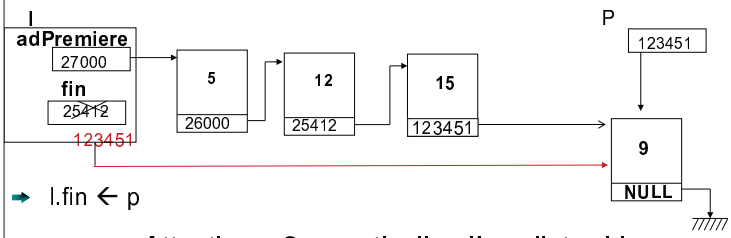
- Chaînage en queue :

P ← reserve Cellule



Autre méthode de création d'une liste

- p ↑.info ← 9
- p ↑.suivant ← null
- (l.fin) ↑.suivant ← p

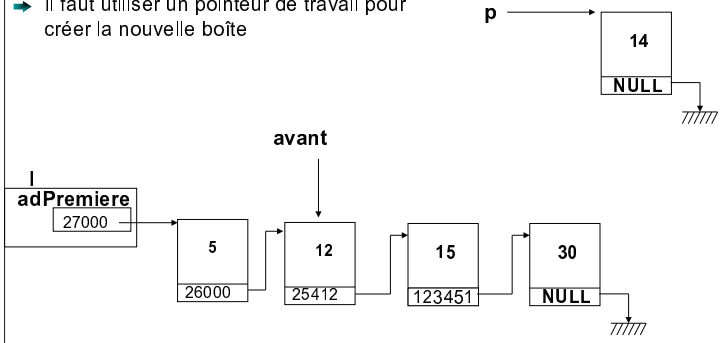


- l.fin ← p

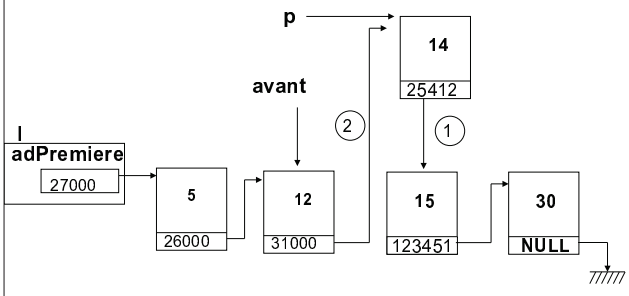
Attention : Cas particulier d'une liste vide

Insertion dans une liste chaînée

- Il faut savoir où insérer
- Il faut utiliser un pointeur de travail pour créer la nouvelle boîte



Insertion dans une liste chaînée(suite)



Problème : Comment trouver la cellule pointée par avant ?

Coût des opérations

	Ajout en tete	Ajout en fin	insertion	Recherche
Tableaux	lineaire			Log(n)
Listes chaînées	constant	?	?	linéaire