Type Abstrait de Données (TAD):
Un Aperçu

Pour définir un TAD on se donne

- une **notation** qui décrit les données, avec des noms de sortes (qui seront des types en programmation), par exemple: Booléen, Entier, Liste,
- des **opérations** applicables à ces données (ou primitives), avec leurs profils, Par exemple, l'opération longueur a pour profil

  Longueur : Liste $\rightarrow$ Entier
- les **propriétés** de ces opérations (ou sémantique)

Un exemple de TAD:

Le type **entier** :

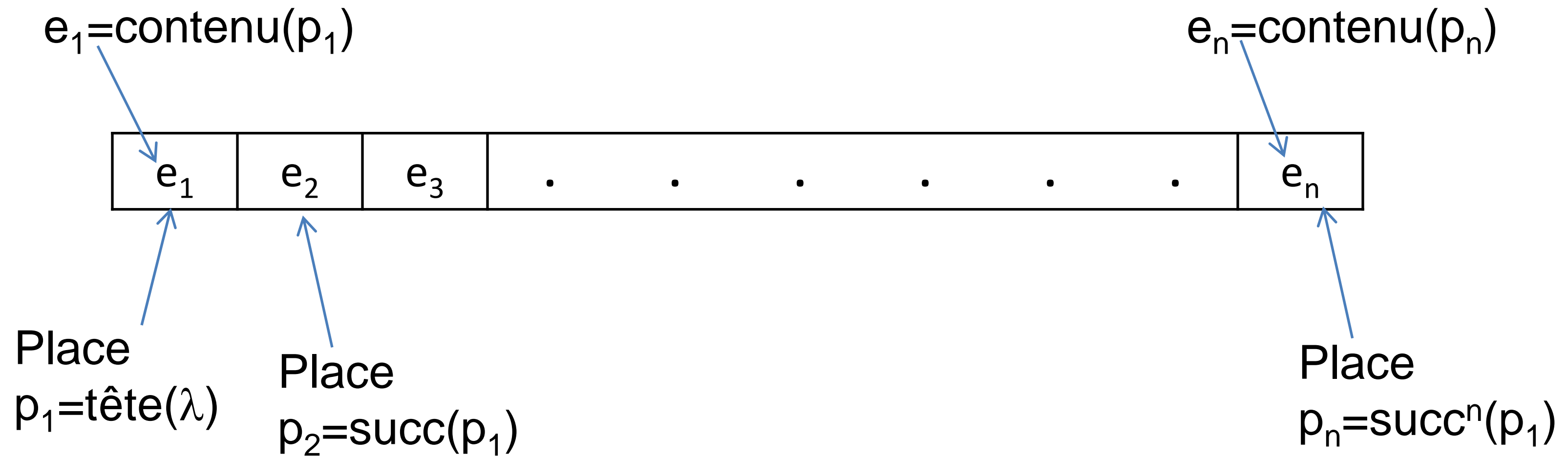
- Chiffres décimaux, précédés de + ou – (**Notation**)
- **Opérations** arithmétiques : +, -, *, /
- Sens usuel de ces opérations (**Propriétés**)

Chapitre 3

Structures séquentielles :

Listes, Piles et Files

Une liste $\lambda=<e_1, e_2, e_3, \ldots e_n>$



n=longueur(λ)

Si n=0, alors λ=liste-vide

Changement par **insertion** et **suppression** des éléments.

Deux cas particuliers de listes :

les piles : insertion et suppression en tête

les files : insertion en queue et suppression en tête.

Schéma de traitement classique d’une liste par un algo :

```
x=tête(λ);
traiter(contenu(x));
for(int  i=1; i<=longueur (λ)-1; i++)
    {       x=succ(x) ;
            traiter(contenu(x));
    }
```

Opérations de base sur une liste:

- accéder au **i**ème élément
- Supprimer le **i**ème élément
- Insérer un élément à la **i**ème place.

TAD liste itérative

Sortes définies

Sortes prédéfinies

Sorte Liste

Utilise Entier, Elément

Opérations

Observateur de Liste

liste-vide	:		→ liste
ième	:	Liste x Entier	→ Elément
longueur	:	Liste	→ Entier
supprimer	:	Liste x Entier	→ Liste
insérer	:	Liste x Entier x Elément	→ Liste

Op.internes pour Liste

Pré-conditions : où λ : Liste; k : Entier

ieme (λ, k)	est-défini-ssi	$1 \leq k \leq$ longueur(λ)
supprimer (λ, k)	est-déf-ssi	$1 \leq k \leq$ longueur(λ)
insérer(λ, k, e)	est-déf-ssi	$1 \leq k \leq$ longueur(λ)+1

K=longueur(λ)+1 correspond à l'insertion en fin de liste.

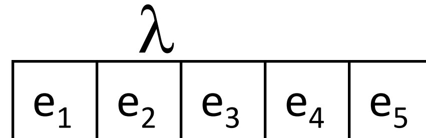
Axiomes : où λ : Liste; k, i : Entier; e: Elément

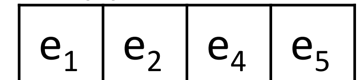
Axiomes pour Longueur :

Longueur(liste-vide)=0

$\lambda \neq$ liste-vide & $1 \leq k \leq$ longueur(λ)

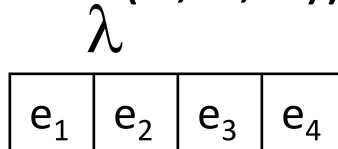
$\Rightarrow$ longueur(supprimer(λ, k))=longueur(λ)-1

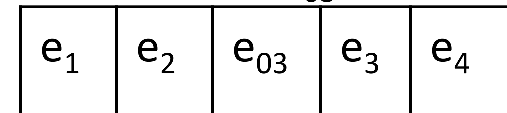




$1 \leq k \leq$ longueur(λ)+1

$\Rightarrow$ longueur(insérer(λ, k, e))=longueur (λ)+1



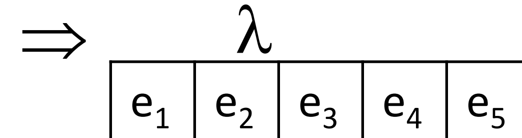


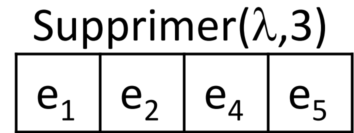
L'observateur ième caractérise le contenu de la ième place.

Exemple : ième(λ, 3) = e_3

Axiomes pour ième:

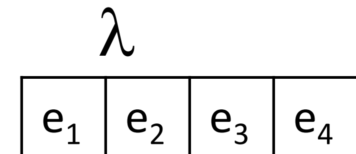
$\lambda \neq$ liste-vide & $1 \leq k \leq$ longueur(λ) & $1 \leq i < k \Rightarrow$
ième(supprimer(λ, k), i)=ième(λ, i)



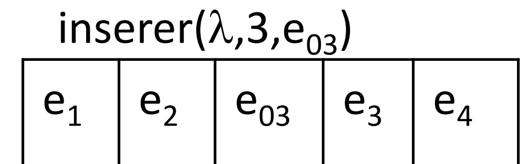


$\lambda \neq$ liste-vide & $1 \leq k \leq$ longueur(λ) &
$k \leq i \leq$ longueur(λ)-1 $\Rightarrow$ ième(supprimer(λ, k), i)=ième(λ, i+1)

$1 \leq k \leq$ longueur(λ)+1 & $1 \leq i < k \Rightarrow$
ième(insérer(λ, k, e), i)=ième(λ, i)



$1 \leq k \leq$ longueur(λ)+1 & k=i $\Rightarrow$
ième(insérer(λ, k, e), i)=e



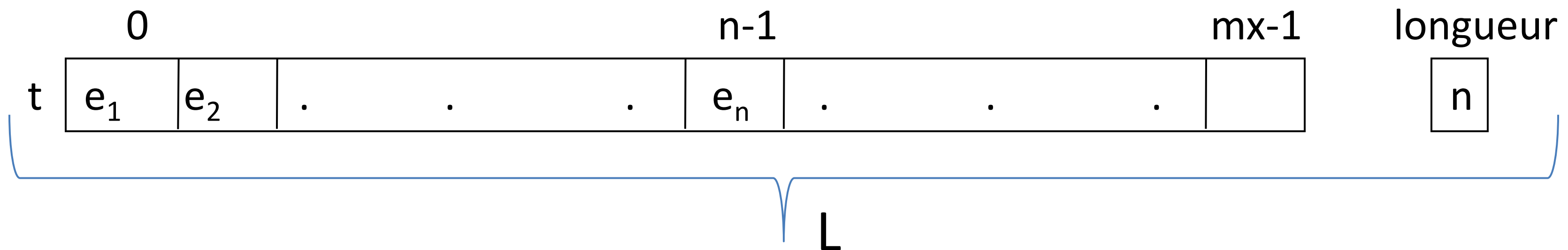
$1 \leq k \leq$ longueur(λ)+1 & $k < i \leq$ longueur(λ)+1 $\Rightarrow$
ième(insérer(λ, k, e), i)=ième(λ, i-1)

ième n'est pas défini pour les listes vides.

Représentation des listes

a) Représentation contigüe: une liste est un couple <tableau, entier> où tableau est surdimensionné (majoré).

```
const int mx=90;
struct Tliste   {  Elément  t[mx] ;
                   int longueur;}
```



Soit L: Tliste;

Si L est vide alors L.longueur=0

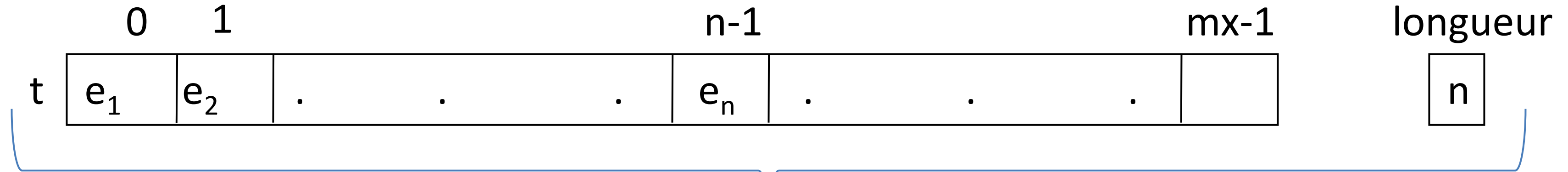
ieme(L, i) = L.t[i-1]

TD N° 03

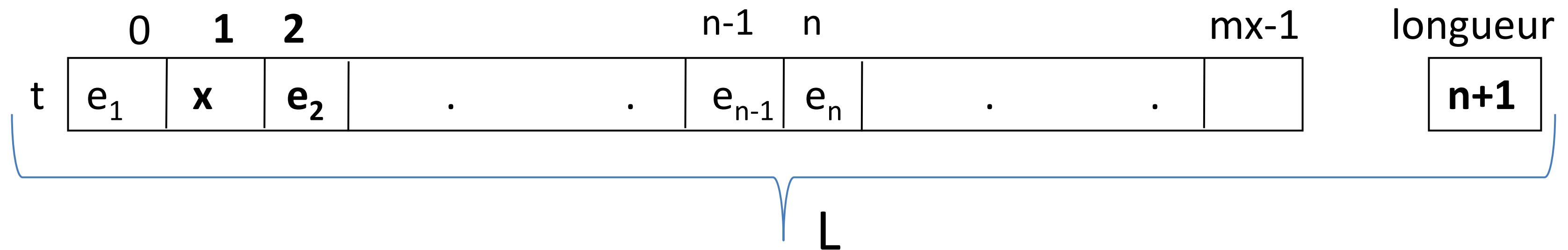
Exercice 01: Programmer les opérations du TAD liste itérative en utilisant la représentation contigüe.

Principe d'insertion d'un élément:

Avant :

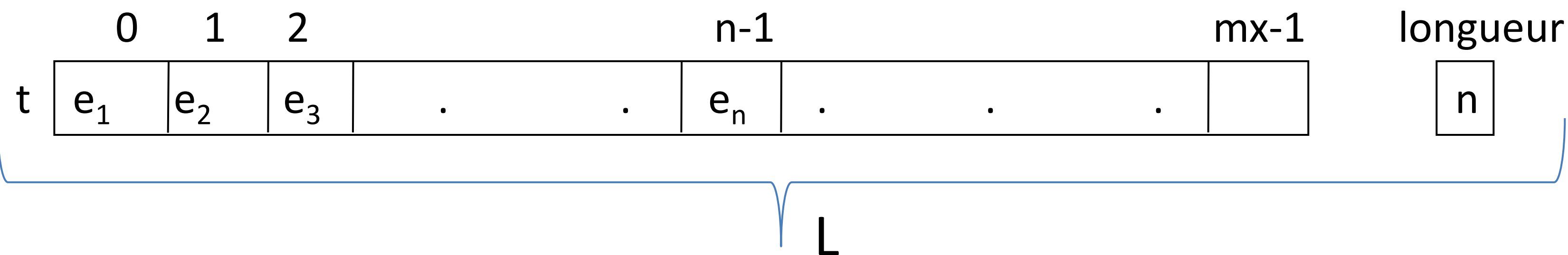


Après inserer(L, 2, x) :

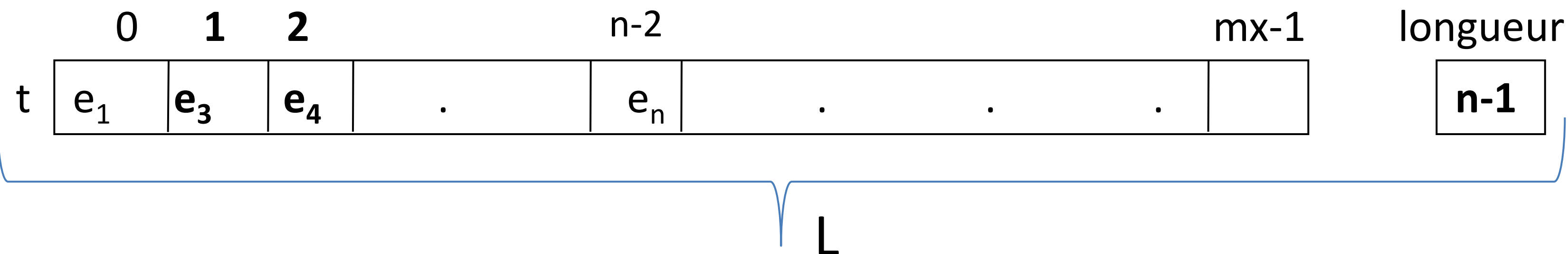


Principe de suppression d'un élément:

Avant



Après supprimer(L, 2) :



Avantages :

- Accès direct aux éléments
- Parcours séquentiel simple

Inconvénients:

- Nécessité de décalage pour insertion et suppression

- Nécessité de majoration de la taille de la liste.

- Représentation coûteuse de la liste vide.

Remarques :

- Une suppression demande, au pire, n-1 affectations (suppression du 1er élément).
- Une insertion demande, au pire, n+1 affectations (insertion à la 1ère place).
- La représentation contigüe est bien adaptée aux listes itératives (accès et parcours faciles).
- Cette représentation est mal adaptée aux listes récursives : l’accès au 1er élément est facile, mais il n’est pas commode de représenter les opérations cons et fin.

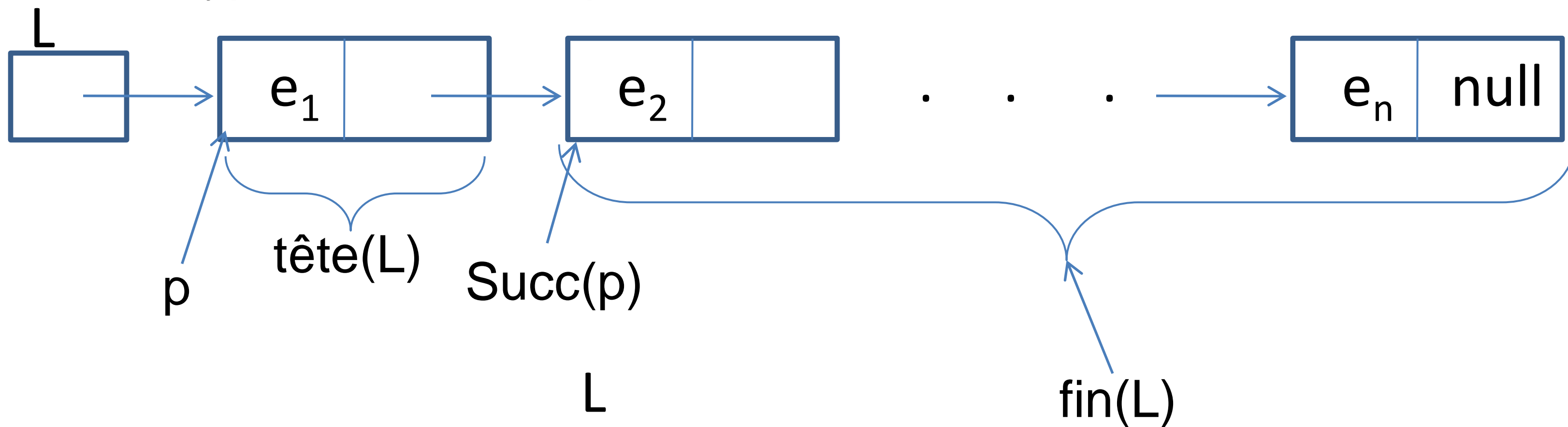
Représentation chaînée

Soit $L=<e_1, e_2, e_3, . . . , e_n >$

En C++ :

```
struct  cellules {  élément val;
                    cellules *lien;
                 };

typedef cellules* pliste;
```

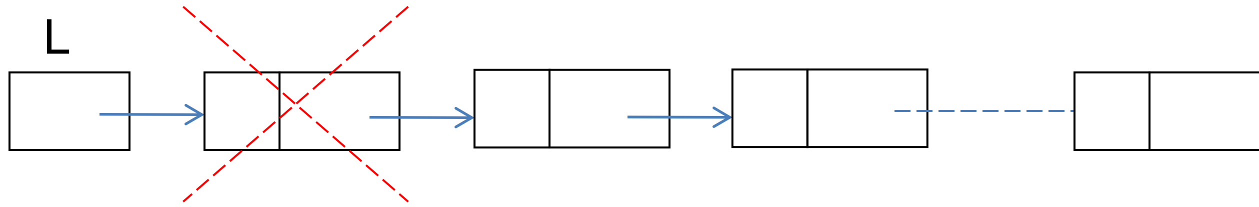


La liste vide : NULL

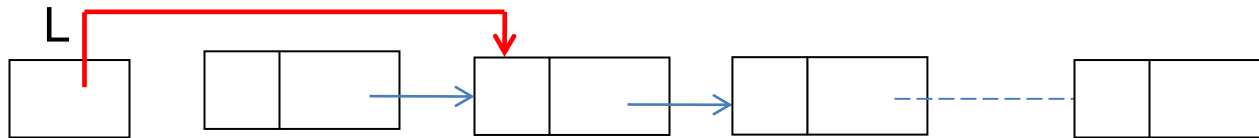
Suite au TD N° 03

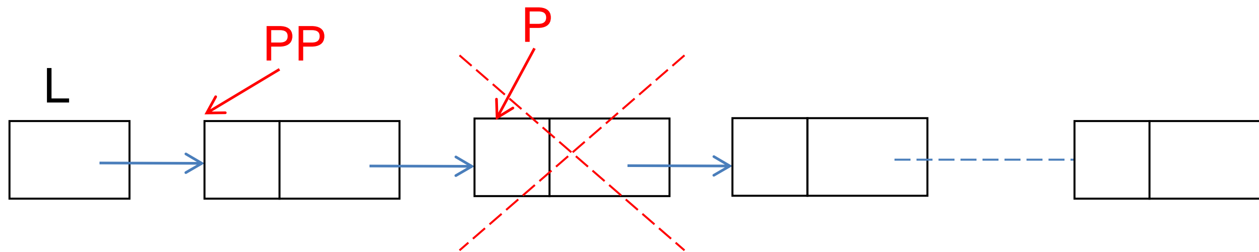
Exercice 02: Programmer les opérations du TAD liste itérative pour la représentation chaînée.

Principe de suppression

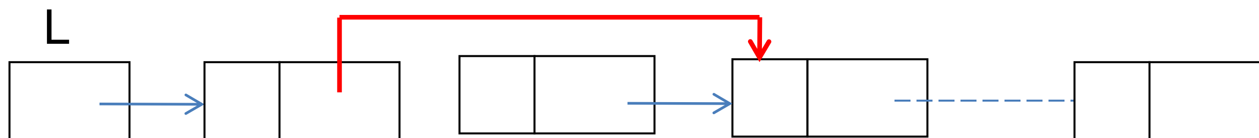


Suppression en tête

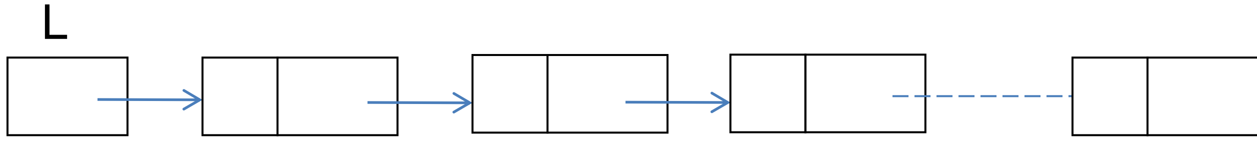




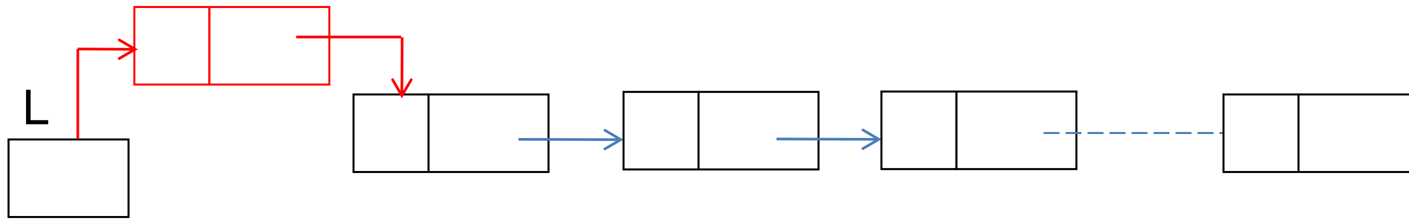
Suppression au milieu

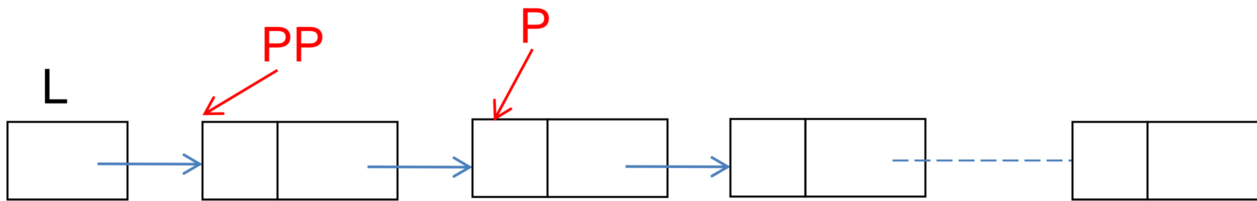


Principe d'insertion

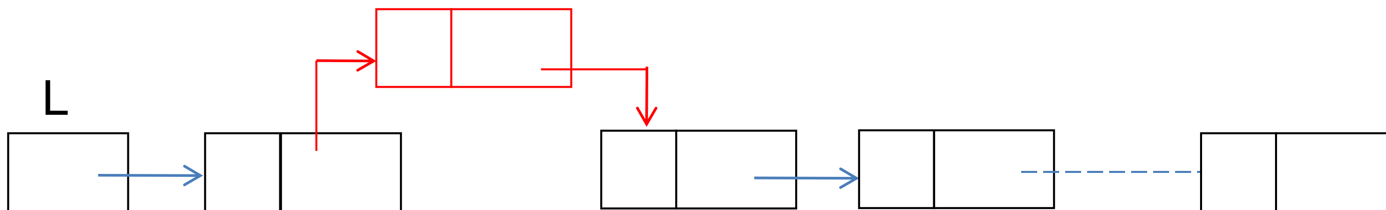


Insertion en tête





Insertion au milieu



Inconvénients de la représ. chaînée :

- Place mémoire supplémentaire
- Accès séquentiel au kème élément; il faut passer par les k-1 éléments qui le précèdent.
- Calcul de la longueur nécessite le parcours de toute la liste.

Avantages :

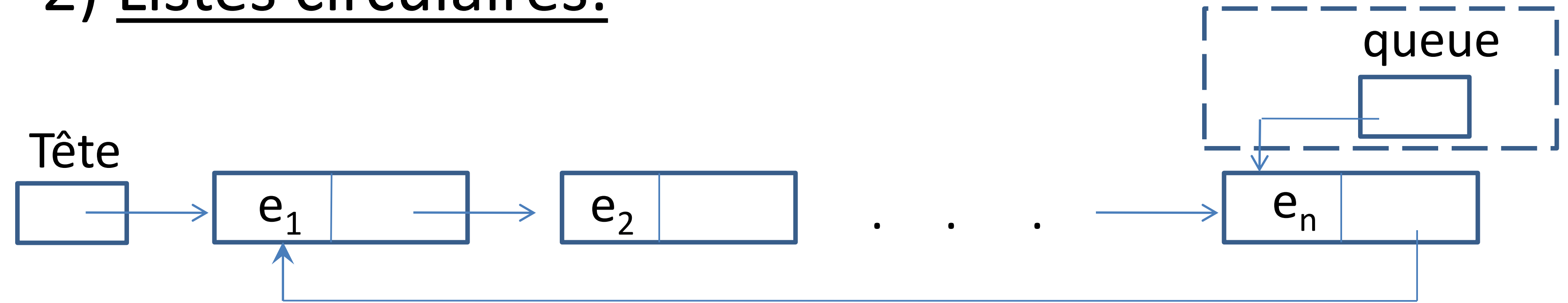
- Pas de longueur max (majoration) sur la taille.
- Pas de décalage pour insertion et suppression.

Variantes des listes chaînées

1)

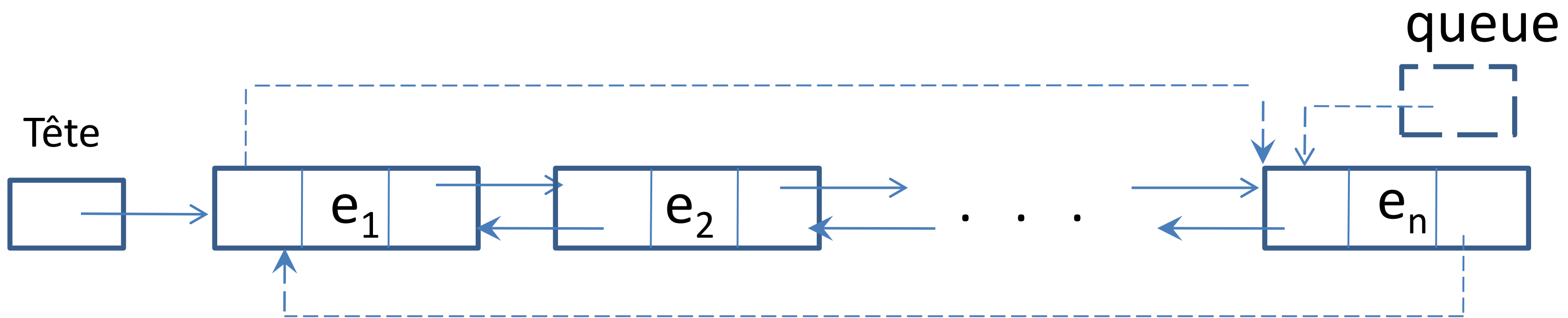
Avantage : éviter le traitement spécial pour l'insertion et la suppression en tête.

2) Listes circulaires:



Avantage: gestion des files (plus tard).

3) Listes bilatères ou doublement chaînées



Suite au TD N° 03

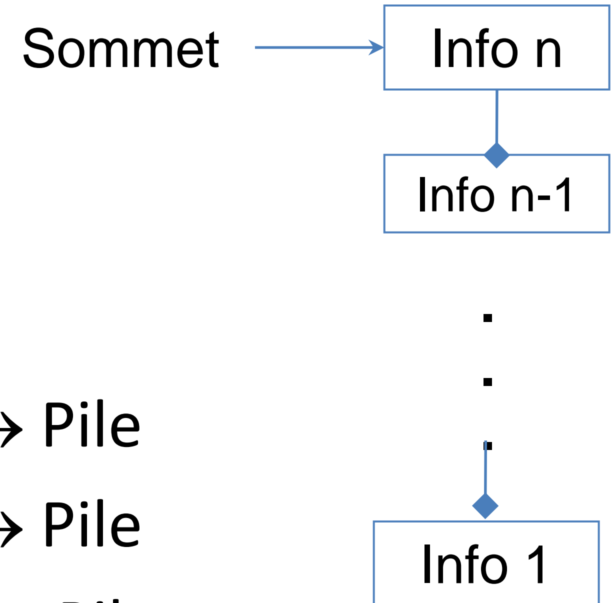
Exercice 03: Programmer les opérations du TAD liste itérative pour la représentation chaînée avec tête de liste.

Exercice 04: Programmer les opérations du TAD liste itérative pour la représentation par listes circulaires.

Exercice 05: Programmer les opérations du TAD liste itérative pour la représentation par listes bilatères.

Les Piles

Dans une pile : l'insertion(**empiler**) et la suppression(**dépiler**) se font à une seule extrémité : le sommet de pile. On parle de structures LIFO : Last In First Out.



Signature du TAD pile:

Sorte Pile

Utilise Booléen, Elément

Opérations pile-vide : → Pile

empiler : Pile x Elément → Pile

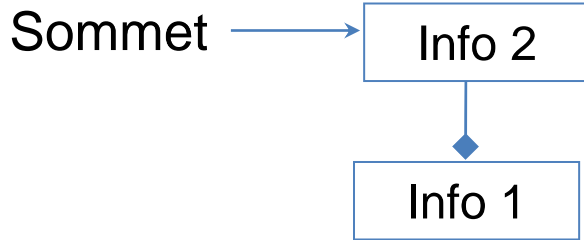
dépiler : Pile → Pile

sommet : Pile → Elément

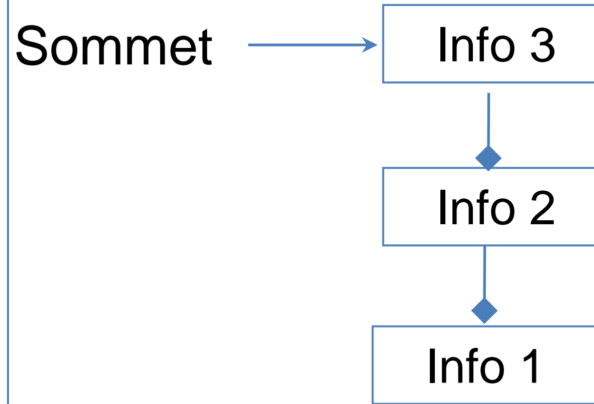
est-vide : Pile → Booléen

Empilement

Avant l'empilement de info 3 :

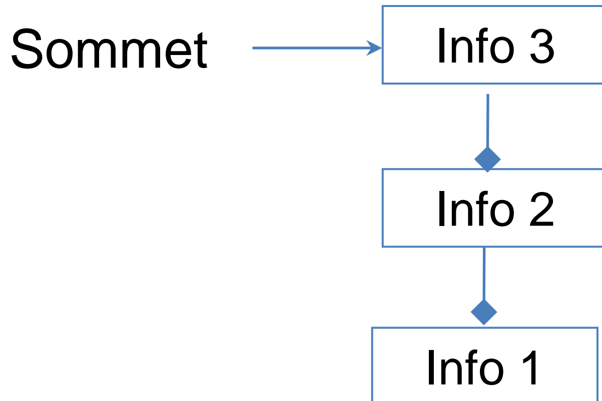


Après l'empilement de info 3 :

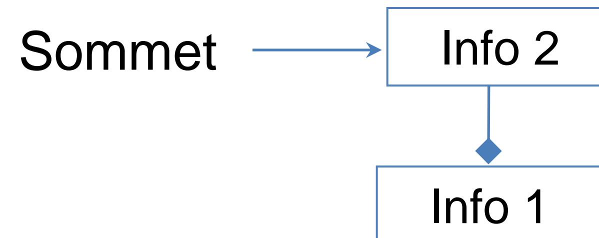


Dépilement

Avant dépilement de info 3 :



Après dépilement de info 3 :



Pré-conditions où p: Pile

dépiler (p) est-déf-ssi est-vide(p) = faux

sommet (p) est-déf-ssi est-vide (p) = faux

Axiomes où p: Pile; e : Elément

est-vide (pile-vide) = vrai

est-vide (empiler(p, e)) =faux

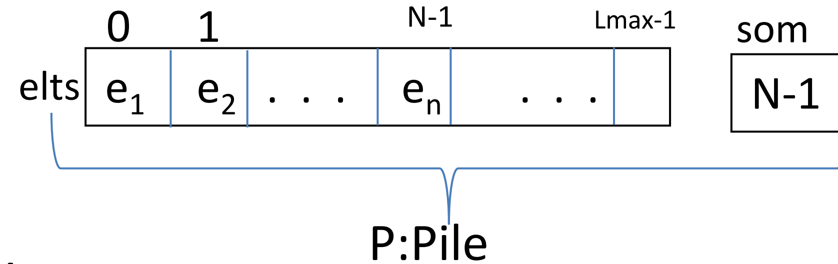
dépiler (empiler (p, e)) =p

sommet (empiler (p, e)) =e

Représentation des piles

a) Représentation contigüe:

```
struct pile{
        int som;
        Element elts[lmax] ; };
```



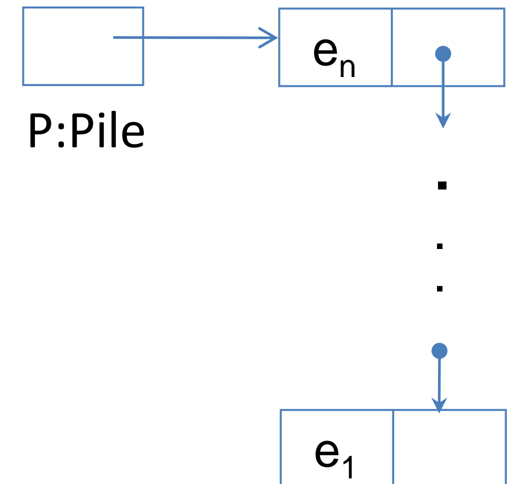
Où som est l'indice du sommet de la pile.

La programmation des opérations du TAD Pile sera abordée en TD.

b) Représentation chaînée : Les différents éléments sont chaînés entre eux. Le sommet est représenté par le 1er élément de la liste (dernier inséré).

La pile vide est donnée par le pointeur NULL.

```
struct Elt {  Element val ;
              Elt* lien ;
           }
 typedef Elt* Pile;
```



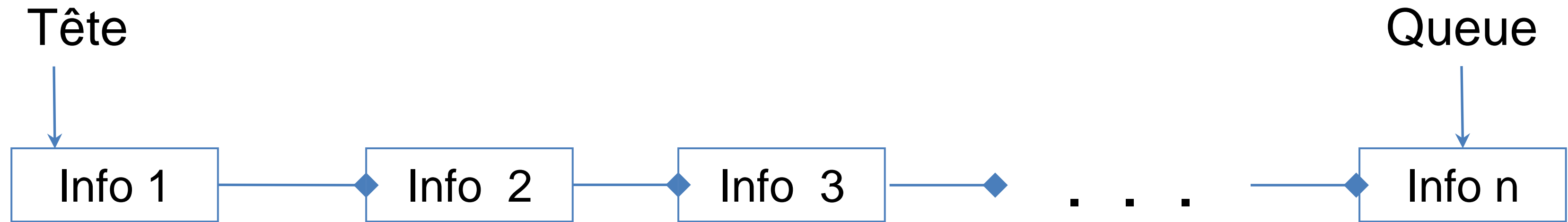
La programmation des opérations du TAD Pile sera abordée en TD.

Suite au TD N° 03

Exercice 06: Programmer les opérations du TAD pile pour les deux représentations contigüe et chaînée.

Les files

- insertions (ajouts) à une extrémité (queue).
- suppressions(retraits) à l’autre extrémité (tête).
- Analogie avec les files d’attente : FIFO ou First In First Out.



Signature du TAD file:

Sorte File

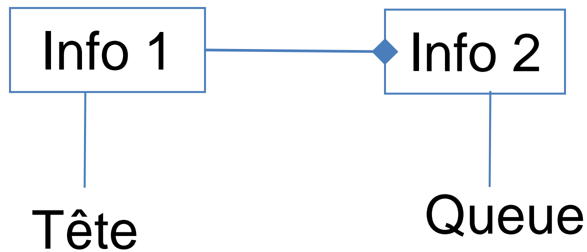
Utilise Booléen, Elément

Opérations

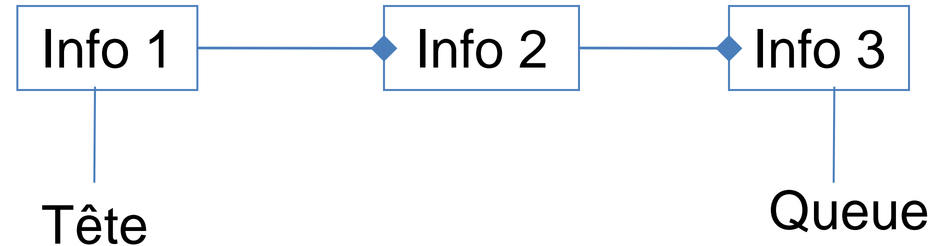
file-vide :	→ File
ajouter : File x Elément	→ File {enfiler}
retirer : File	→ File {défiler}
premier : File	→ Elément
est-vide : File	→ Booléen

Ajout ou Enfilement

Avant l'enfilement de info 3 :

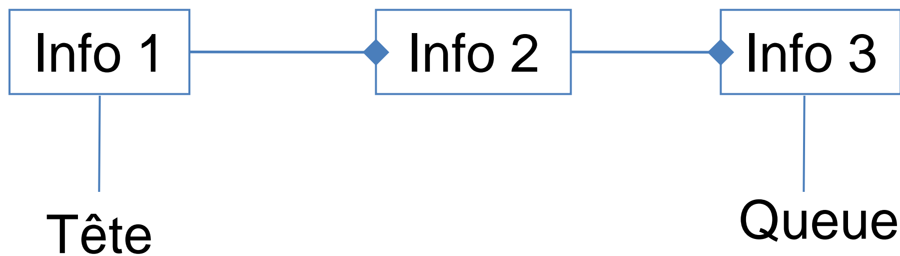


Après l'enfilement de info 3 :

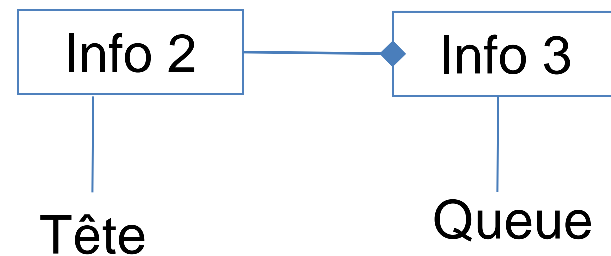


Retrait ou Défilement

Avant défilement de info 1 :



Après défilement de info 1 :



Soient f: file e: Elément

Pré-conditions

premier (f) est-déf-ssi est-vide (f) = faux

retirer (f) est-déf-ssi est-vide (f)= faux

Axiomes

est-vide(f)=vrai $\Rightarrow$ premier(ajouter(f, e))=e

est-vide(f)=faux $\Rightarrow$ premier(ajouter(f, e))=premier(f)

est-vide(f)=vrai $\Rightarrow$ retirer(ajouter(f, e))=file-vide

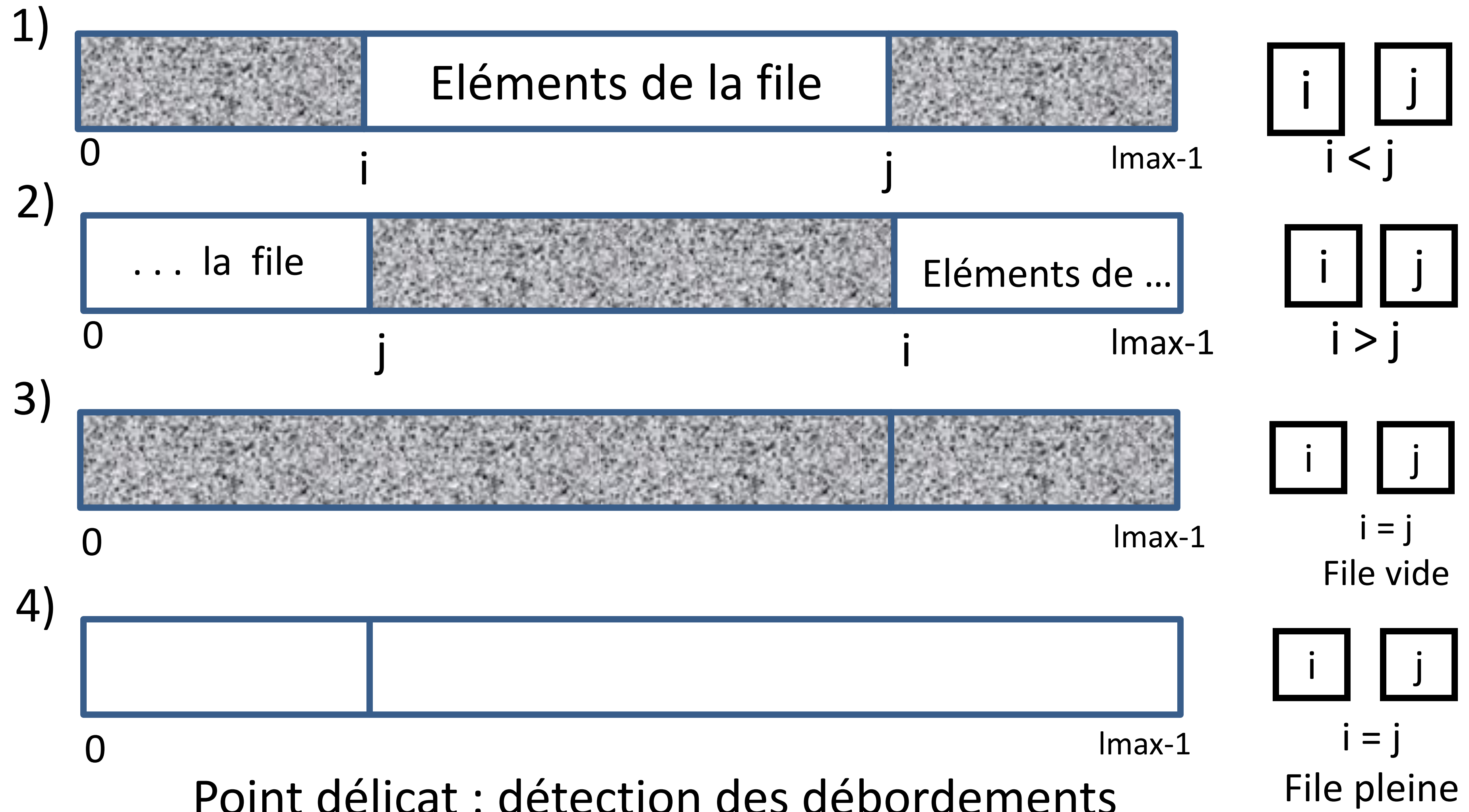
est-vide(f)=faux $\Rightarrow$ retirer(ajouter(f, e))=ajouter(retirer(f), e)

est-vide(file-vide)=vrai

est-vide(ajouter(f,e))=faux

Représentation des files

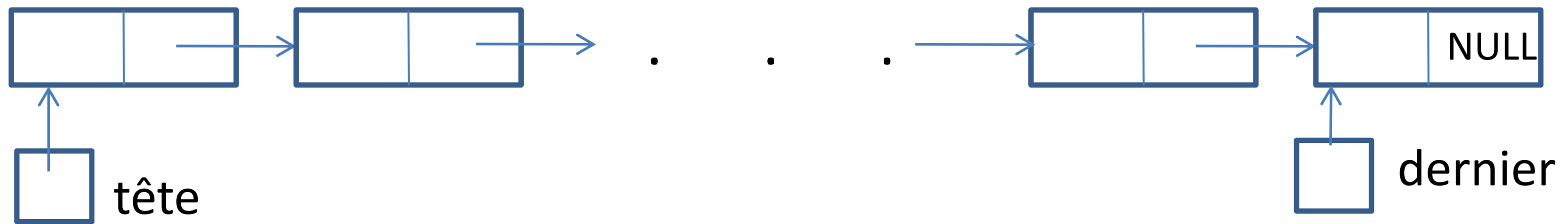
- Contigüe : i : indice du 1er élément (i initial = -1/0)
  j : indice de la 1ère place libre (j initial = 0/-1).



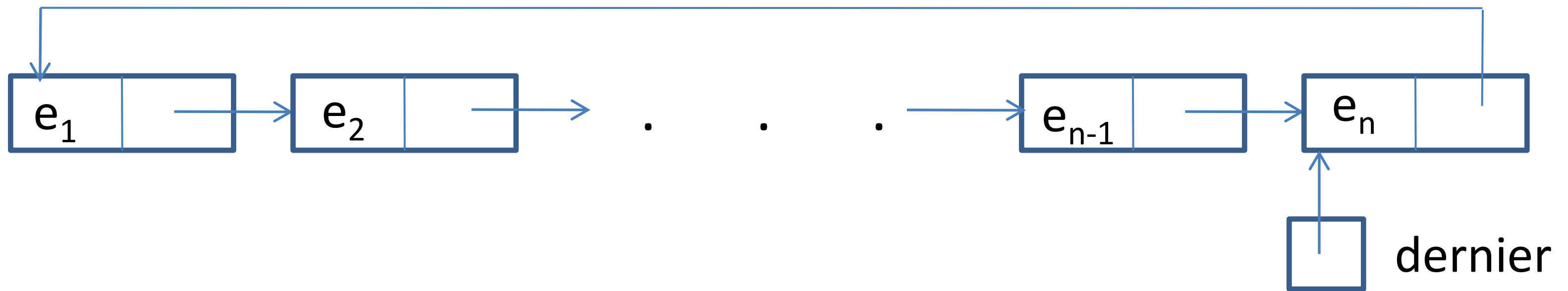
Point délicat : détection des débordements

- Chaînée : Il y a plusieurs possibilités

1)



2)



Suite au TD N° 03

Exercice 07: Programmer les opérations du TAD file pour les deux représentations contigüe et chaînée.

TD N° 03

Exercice 01: Programmer les opérations du TAD liste itérative en utilisant la représentation contigüe.

Exercice 02: Programmer les opérations du TAD liste itérative pour la représentation chaînée.

Exercice 03: Programmer les opérations du TAD liste itérative pour la représentation chaînée avec tête de liste.

Exercice 04: Programmer les opérations du TAD liste itérative pour la représentation par listes circulaires.

Exercice 05: Programmer les opérations du TAD liste itérative pour la représentation par listes bilatères.

Exercice 06: Programmer les opérations du TAD pile pour les deux représentations contigüe et chaînée.

Exercice 07: Programmer les opérations du TAD file pour les deux représentations contigüe et chaînée.