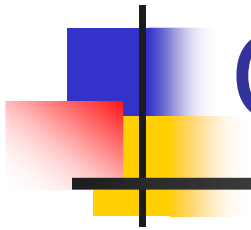
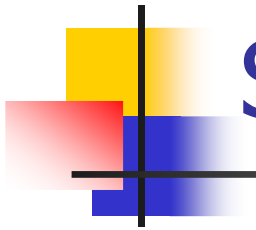


Cours 8 : Liste, Pile et File

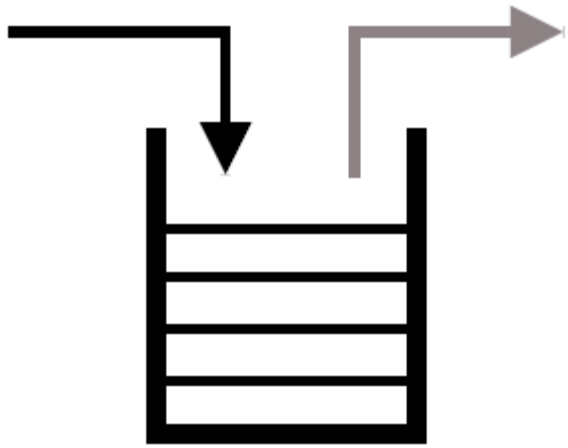


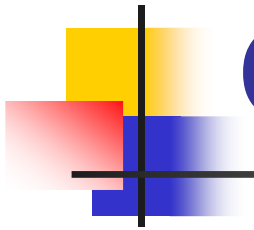
Structures de données linéaires

- Structures de données abstraites et dynamiques
  - Pile
  - File
  - Liste
- Implantation :
  - Utilisation de tableaux
  - Utilisation de pointeurs (liste chaînée)

Pile (stack)

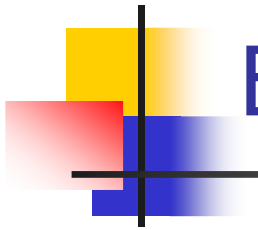
- Last In – First Out (LIFO)
- Principe de la pile d'assiettes





Opérations sur une pile

- Une pile est une liste sur laquelle on autorise seulement 4 opérations :
  1. Tester si la pile est vide
  2. Consulter le dernier élément (=s ommet)de la pile (« peek »)
  3. Empiler un élément, le mettre au sommet de la pile (« push »)
  4. Dépiler un élément du sommet (« pop »)

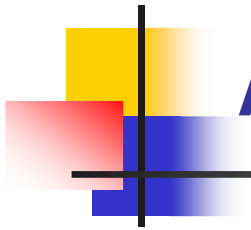


Exemple : compilation

- Analyse syntaxique: Reconnaissance des mots bien parenthésés
  - Accepter les mots comme

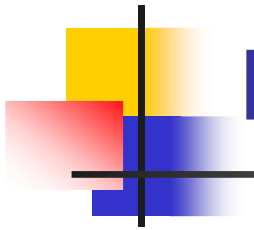
    (), ()() ou ((()())())
  - Rejeter les mots comme

)(, ()(() ou ((()()))



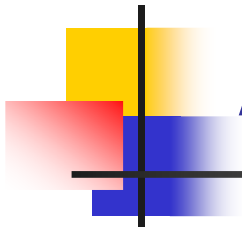
Algorithme

- Nous stockons les parenthèses ouvrantes non encore refermées dans une pile de caractères
- Le programme lit caractère par caractère le mot entré :
  - si c'est une ouvrante, elle est empilée
  - si c'est une fermante, l'ouvrante correspondante est dépilée
- Le mot est accepté si
  - la pile n'est jamais vide à la lecture d'une fermante et si
  - la pile est vide lorsque le mot a été lu



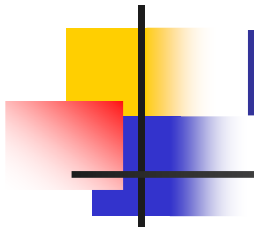
Exemple : notation polonaise

- Notation postfixée des expressions arithmétiques :
  - PAS DE PARENTHÈSES !
  - valeurs
  - opérateurs binaires : +, -, *, /
  - opérateurs unaires : -, sqrt, sin, cos, exp, …
- Infixe à postfixe :
  - 2+5*6 => 256*+
  - (3+5)*2 => 35+2*



Algorithme

Initialiser la pile à vide
Tantque (ce n'est pas la fin de l'expression postfixée)
    prendre l'item suivant de l'expression postfixée
    Si (item est une valeur) **Alors**
        empiler
    Sinon
        Si (item est un operateur binaire) **Alors**
            dépiler dans op2
            dépiler dans op1
            effectuer op1 operateur op2
            empiler le résultat obtenu
        Sinon (item est opérateur unaire)
            dépiler dans op
            effectuer opérateur(op)
            empiler le résultat obtenu
    ...
FinTantque
La seule valeur qui reste dans la pile est le résultat recherché

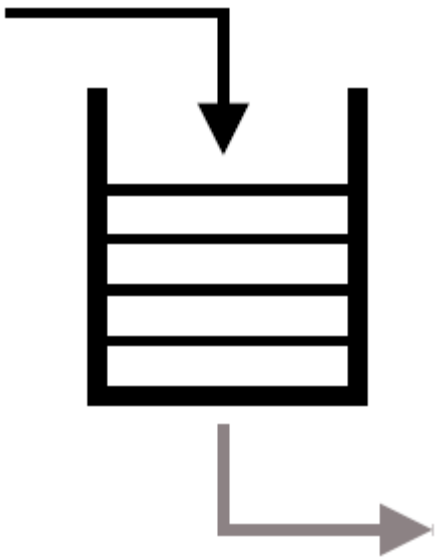


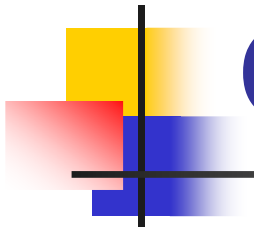
Exemple

6 5 2 3 + 8 * + 3 + *

File (queue)

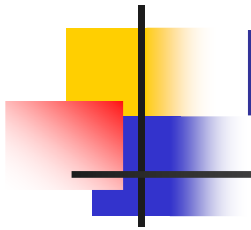
- First In – First Out (FIFO)
- Principe de la file d'attente





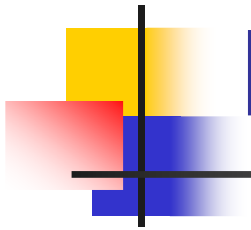
Opérations sur une file

- Une file est une liste sur laquelle on autorise seulement 4 opérations :
  1. Tester si la file est vide
  2. Consulter le premier élément de la file
  3. Enfiler un nouvel élément : le placer en dernier
  4. Défiler un élément : supprimer le premier



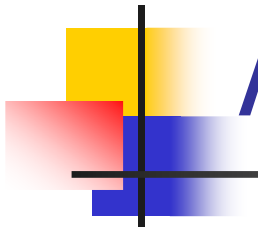
Exemples d’utilisation

- Impression de programmes
  - maintenir une file de programmes en attente d'impression
- Ordonnanceur dans le système d'exploitation
  - maintenir une file de processus en attente d'un temps machine



Liste : opérations

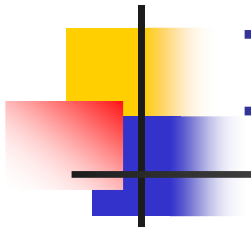
1. Créer une liste vide, tester si une liste est vide
2. Afficher les éléments d'une liste
3. Ajouter un élément à la liste
   a) ajouter en début de liste (tête de liste)
   b) ajouter à la fin de la liste (queue)
   c) ajouter à une position donnée
   d) ajouter après ou avant une position donnée
4. Supprimer un élément d'une liste
   a) supprimer en début de liste
   b) supprimer en fin de liste
   c) supprimer à une position donnée
   d) supprimer avant ou après une position donnée



Autres opérations

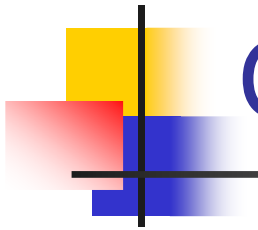
5. Rechercher le i-ème élément
6. Rechercher un élément d'une valeur particulière
7. Trier les éléments d'une liste
8. Fusionner 2 listes
9. ...

En fonction des opérations fréquemment utilisées, choisir une représentation interne pour laquelle les listes sont efficaces



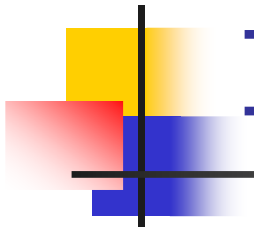
Implantation par tableau

- Avantages :
  - Facile et direct
- Inconvénients :
  - Il faut prévoir une taille maximum pour le tableau
    - Surdimensionné
    - Gaspillage de mémoire



Complexité

- La recherche du k-ème élément se fait en O(1)
- L'opération d'affichage des éléments d'une liste et de recherche d'un élément se font en O(n) - temps linéaire
- L'insertion et la suppression d'éléments sont plus coûteuses (l'insertion à la position 1 nécessite le déplacement de tous les éléments de la liste pour libérer cette position)
- Par conséquent la création d'une liste à l'aide de n insertions successives à partir de la position 1 nécessite un temps en $O(n^2)$ – complexité quadratique

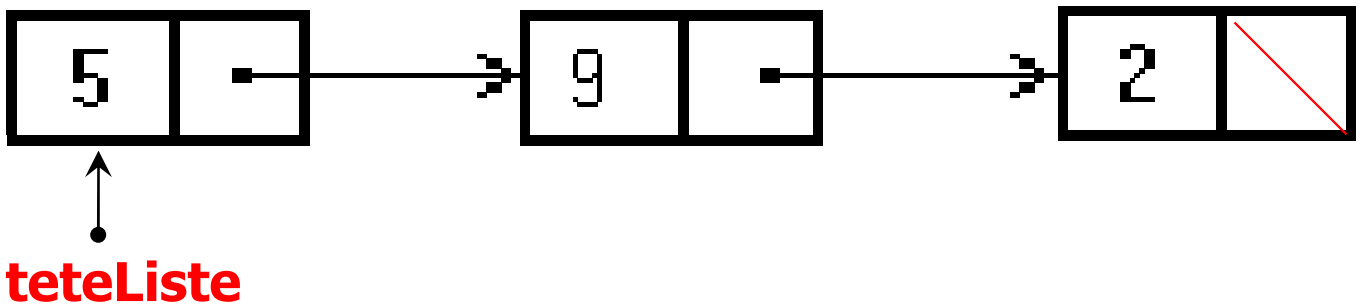


Implantation par liste chaînée

- Pointeur
  - est une variable dont le contenu est une adresse ,
  - cette adresse permet de référencer à l'emplacement d'une autre variable dans la mémoire
- Allocation dynamique de mémoire
- Les éléments d'une liste sont chaînés entre eux :
  - Liste chaînée simple
  - Liste doublement chaînée
  - Liste circulaire

Liste chaînée simple

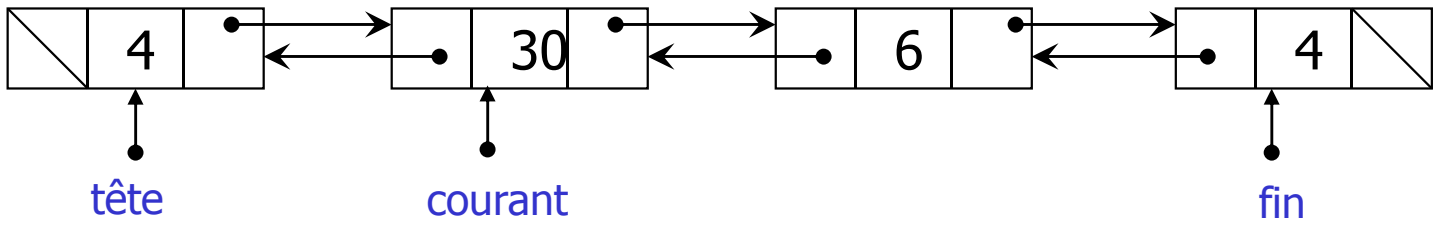
- Chaque nœud contient 2 champs:
  - valeur
  - pointeur qui relie le nœud au nœud suivant



  - dernier nœud : pointeur = NULL

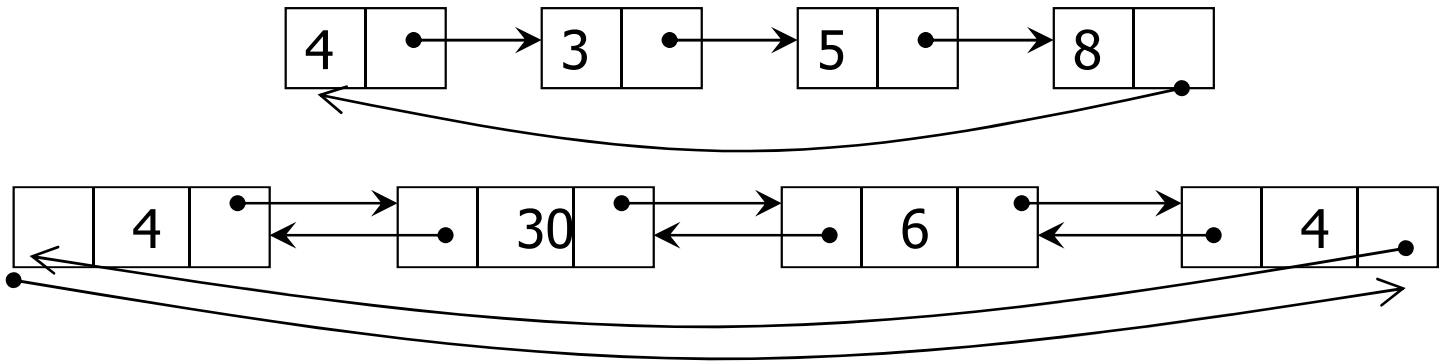
Liste doublement chaînée

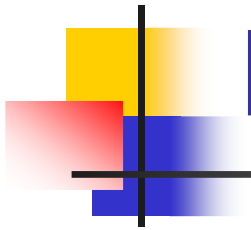
- Un nœud à trois champs :
  1. un champ information
  2. un pointeur vers le successeur
  3. un pointeur vers le prédécesseur



Listes circulaires

- Le pointeur NULL du dernier élément est remplacé par l'adresse du premier élément
- Tous les nœuds sont accessibles à partir de n'importe quel autre nœud
- Une liste circulaire peut être simplement chaînée ou doublement chaînée.



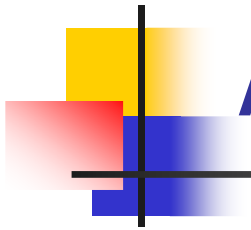


Liste chaînée simple

- Quatre opérations primitives
  - p ← creernoeud() : créer un espace mémoire pour un nœud, le résultat est un pointeur vers ce noeud
  - valeur(p) : accéder à la partie information d'un nœud pointé par p
  - suivant(p) : pointeur du prochain nœud
  - libérer(p) : libérer un nœud pointé par p

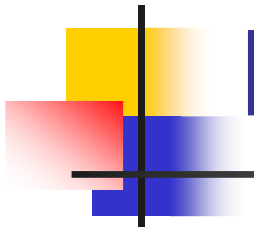
Recherche du k-ième élément

```
Fonction kieme (teteListe : Pointeur,  k : Entier) : T
// 0 <= k < longueur(teteListe)
Variables i : Entier, l : Pointeur
Début
        l ← teteListe
        i ← 0
        Tantque i < k
            l ← suivant(l)
            i ← i + 1
        FinTantQue
        retourner valeur(l)
Fin
```



Affichage

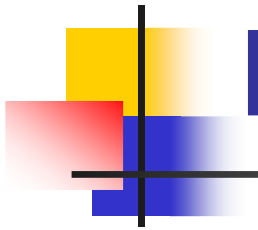
```
Procédure afficher (teteListe : Pointeur)
Variables l : Pointeur
Début
    l ← teteListe
    Tantque l ≠ null
        Ecrire (valeur(l))
        l ← suivant(l)
    FinTantQue
Fin
```



Exercices

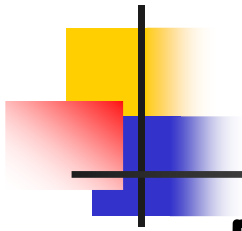
Ecrire une fonction qui renvoie la longueur d'une liste.

Ecrire une fonction qui calcule de nombre d'occurrences d'un élément dans la liste.



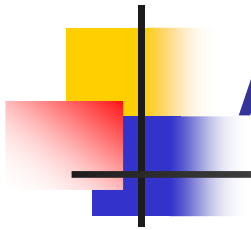
Exercice

```
Fonction longueur (teteListe : Pointeur): Entier
Variables l : Pointeur, n: Entier
Début
    l ← teteListe
    n ← 0
    Tantque l ≠ null
        n ← n + 1
        l ← suivant(l)
    FinTantQue
    Retourner n
Fin
```



Exercice

```
Fonction nbOcc (teteListe : Pointeur, x: T): Entier
Variables l : Pointeur, n: Entier
Début
    l ← teteListe
    n ← 0
    Tantque l ≠ null
        Si valeur(l) = x Alors n ← n + 1 FinSi
        l ← suivant(l)
    FinTantQue
    Retourner n
Fin
```

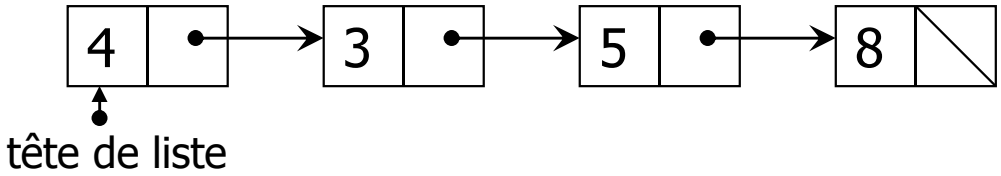


Ajouter

1. ajouter en début de liste (tête de liste)
2. ajouter après une position donnée
3. ajouter avant une position donnée
4. ajouter à la fin de la liste

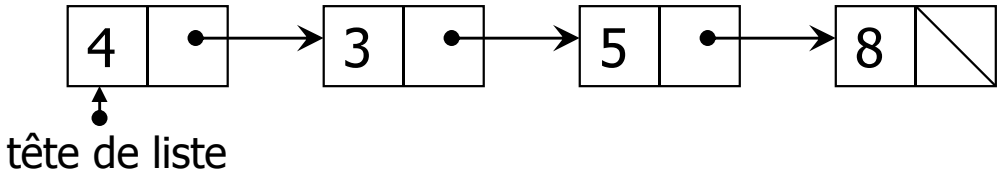
Ajouter en tête de liste

Exemple : ajouter l'élément 6 au début de la liste suivante

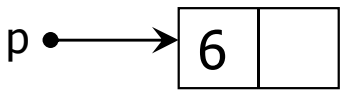


Ajouter en tête de liste

Exemple : ajouter l'élément 6 au début de la liste suivante

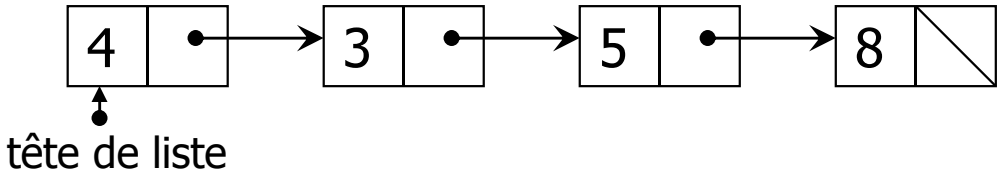


1. Créer le nouveau nœud p et lui donner son contenu

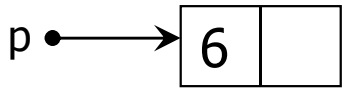


Ajouter en tête de liste

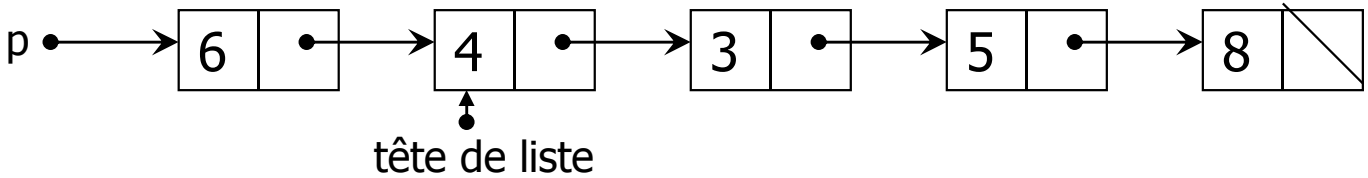
Exemple : ajouter l'élément 6 au début de la liste suivante



1. Créer le nouveau nœud p et lui donner son contenu

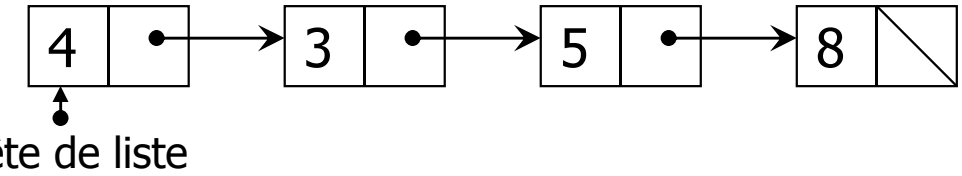


2. Le suivant de p est l'ancienne tête de la liste

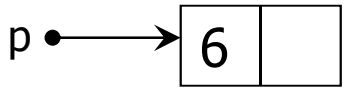


Ajouter en tête de liste

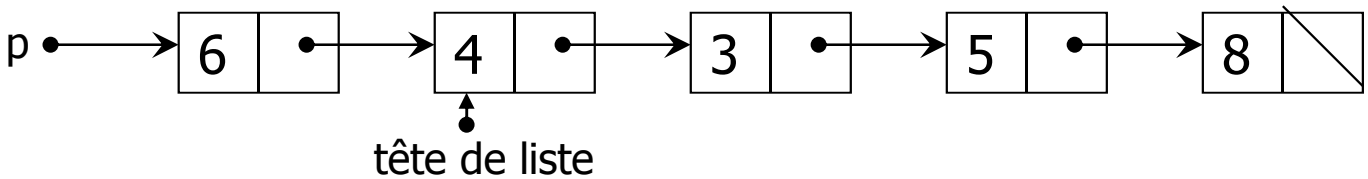
Exemple : ajouter l'élément 6 au début de la liste suivante



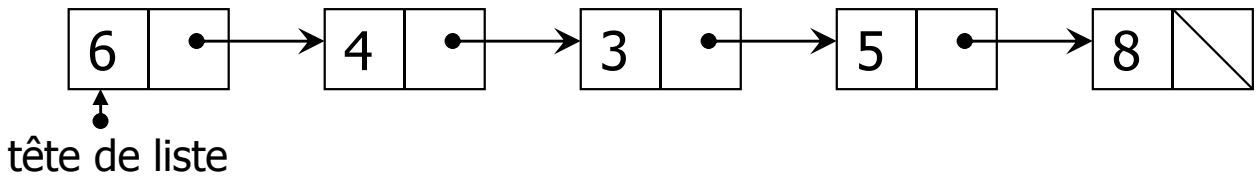
1. Créer le nouveau nœud p et lui donner son contenu

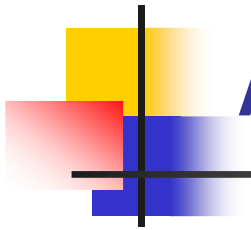


2. Le suivant de p est l'ancienne tête de la liste



3. Le pointeur p devient tête de liste

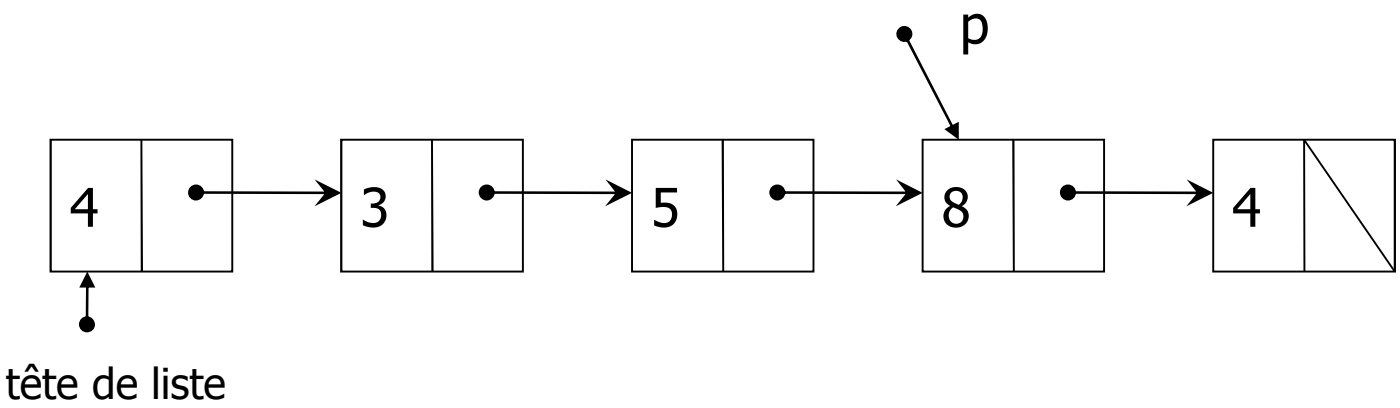




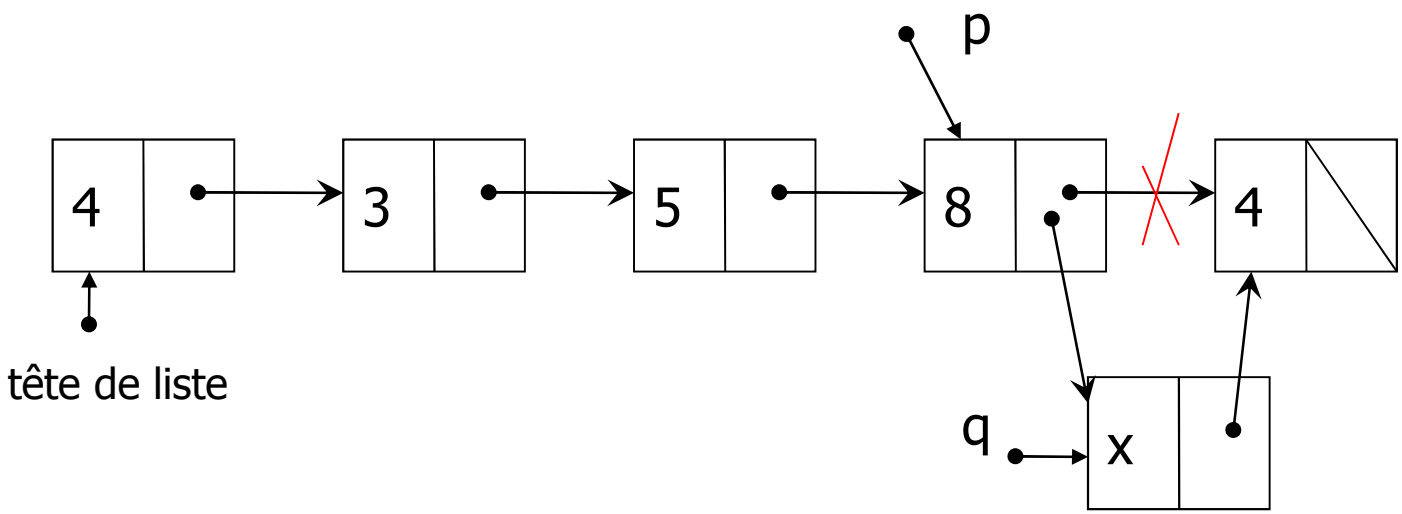
Ajouter en tête de liste

```
Fonction ajoutTete (teteListe : Pointeur, x : T): Pointeur
Variables p : Pointeur
Début
        p ← creernoeud()
        valeur(p) ← x
        suivant(p) ← teteListe
        retourner p // renvoyer la nouvelle tête
Fin
```

Ajouter après un nœud



Ajouter après un nœud



Ajouter après un nœud

Procédure ajouterApres(p : **Pointeur**, x : **T**)
Variables q : **Pointeur**
Début

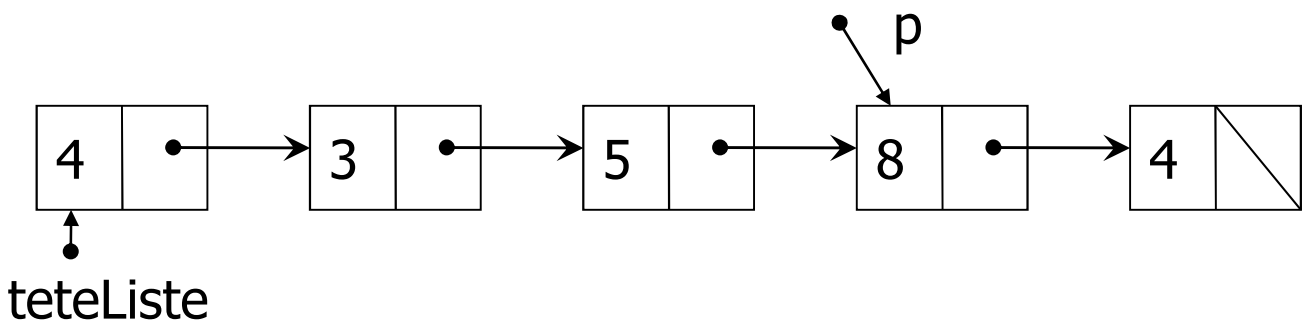
```
q ← creernoeud()
valeur(q) ← x
suivant(q) ← suivant(p)
suivant(p) ← q
```

Fin

Notez: le nombre d'opérations nécessaires pour réaliser cet algorithme est indépendant du nombre d'éléments

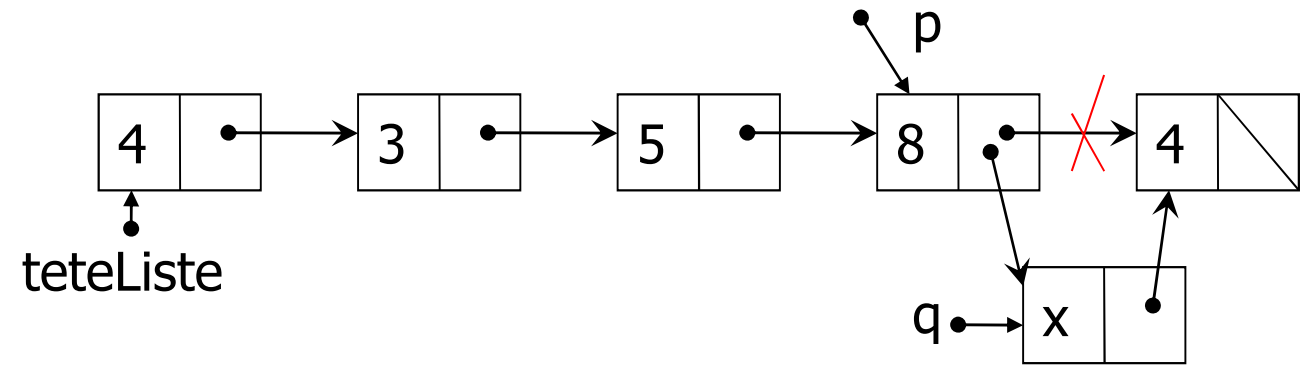
Ajouter avant un nœud

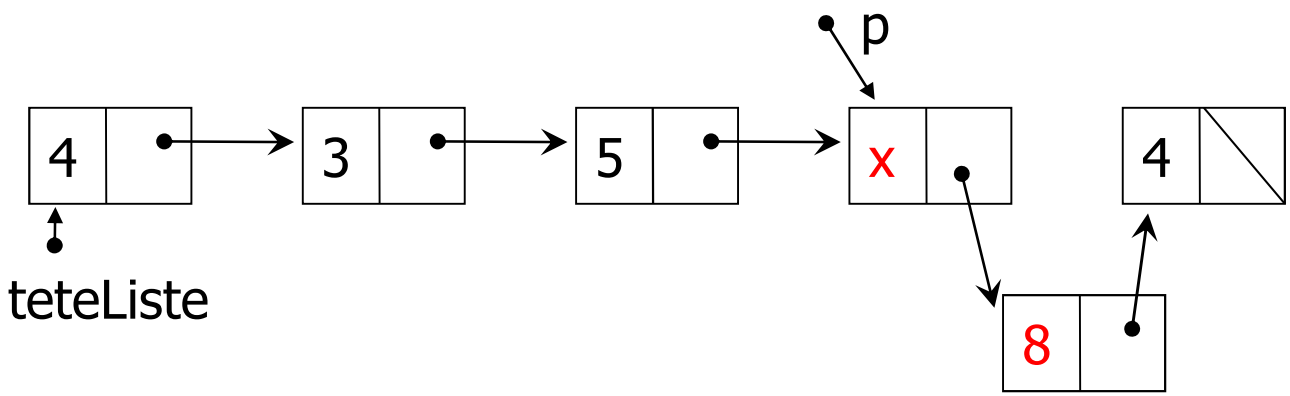
- Pour ajouter un élément avant un nœud, il faut trouver le prédécesseur de p pour établir le lien entre le prédécesseur de p et le nouvel élément



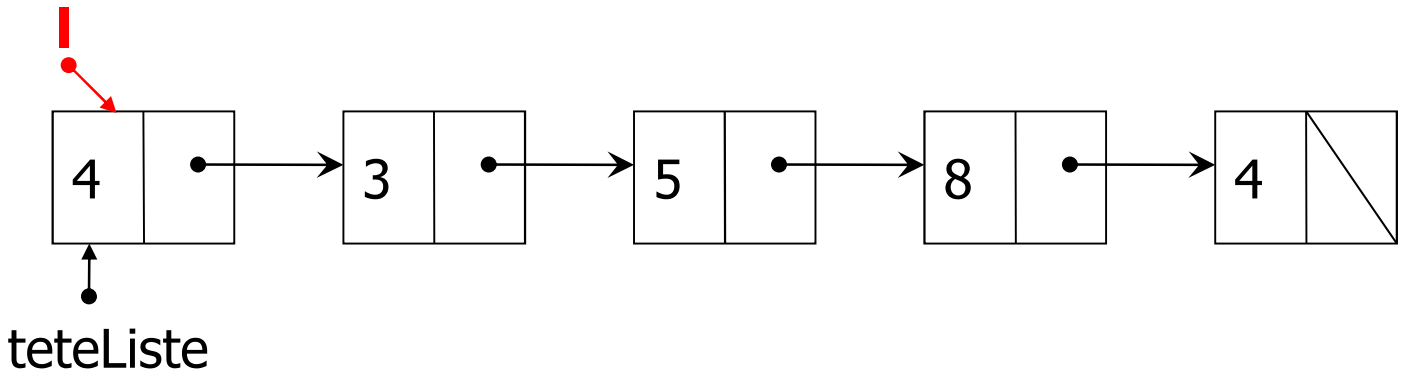
Ajouter avant un nœud

- ASTUCE : ajouter l'élément après le nœud donné puis échanger les valeurs (de p et de q)

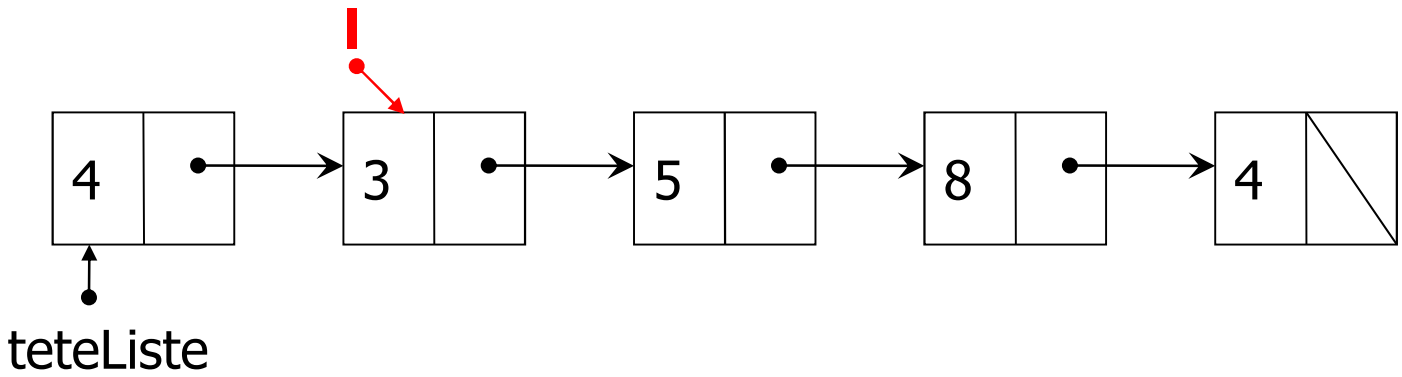




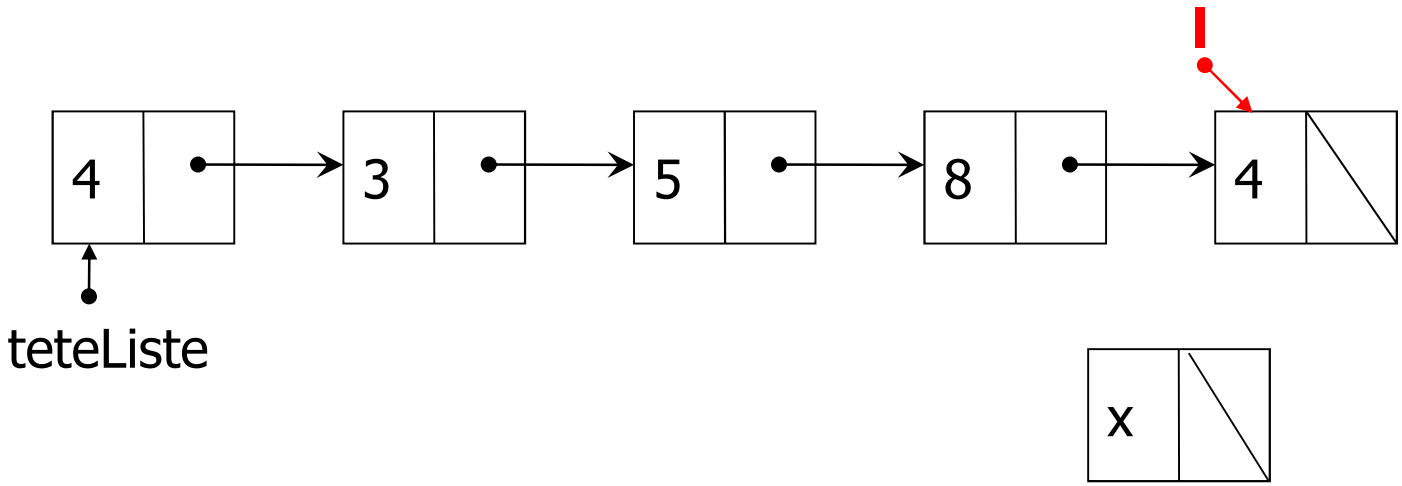
Ajouter en fin de la liste



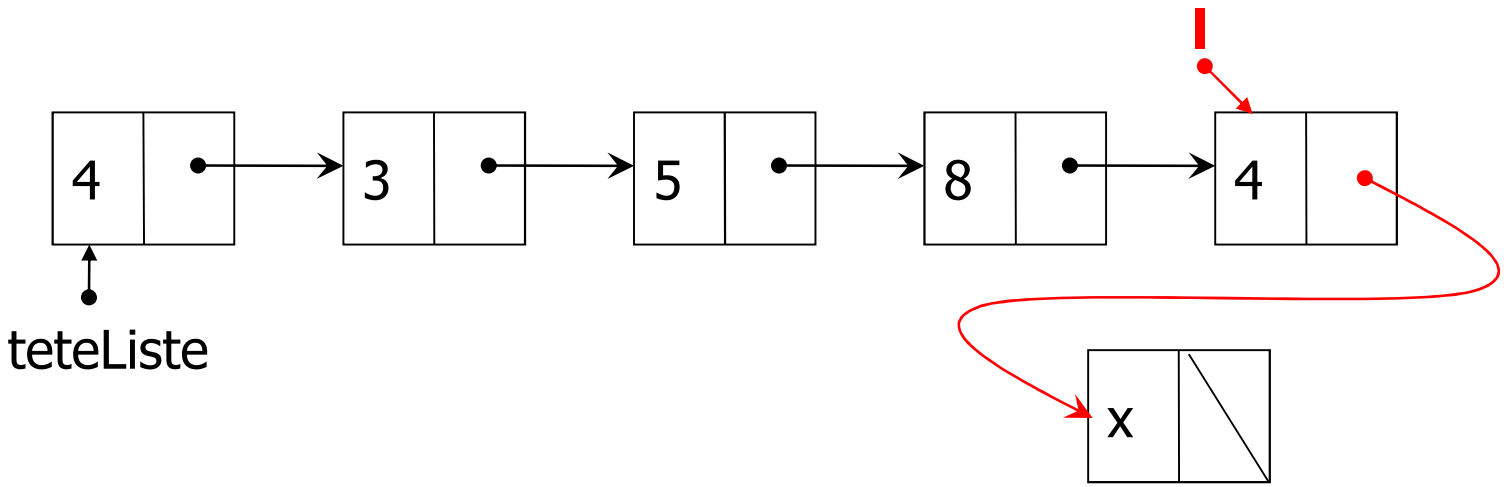
Ajouter en fin de la liste



Ajouter en fin de la liste

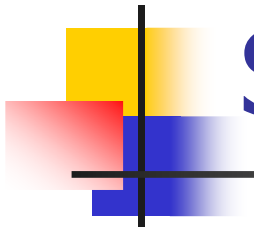


Ajouter en fin de la liste



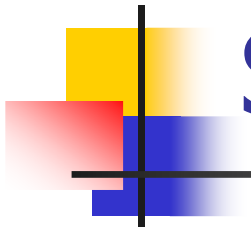
Ajouter en fin de la liste

```
Fonction ajouterFin(teteListe : Pointeur, x : T): Pointeur
Variables l : Pointeur
Début
        Si (teteListe = null) Alors
                Retourner ajouterTete(teteListe,x)
        Sinon
                l ← teteListe     //chercher la fin de liste
                Tantque (suivant(l) ≠ null)
                        l ← suivant(l)
                Fintantque
                ajouterApres(l,x)
                Retourner teteListe
        FinSi
Fin
```



Supprimer

1. supprimer en début de liste
2. supprimer après une position donnée
3. supprimer une position donnée
4. supprimer avant une position donnée
5. supprimer en fin de liste



Supprimer en début de la liste

```
Fonction supprimerTete (teteListe : Pointeur): Pointeur
Variables p : Pointeur
Début
        Si (teteListe ≠ null) Alors
                p ← teteListe
                teteListe ← suivant(teteListe)
                libérer(p)
        FinSi
        Retourner teteListe
Fin
```

Supprimer après un noeud

Procédure supprimerApres(p : **Pointeur**)

Variables q : **Pointeur**

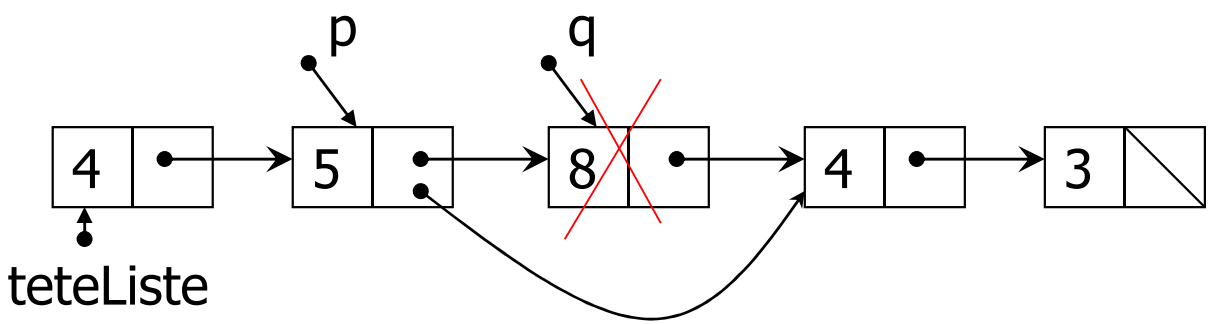
Début

q ← suivant(p)

suivant(p) ← suivant(q)

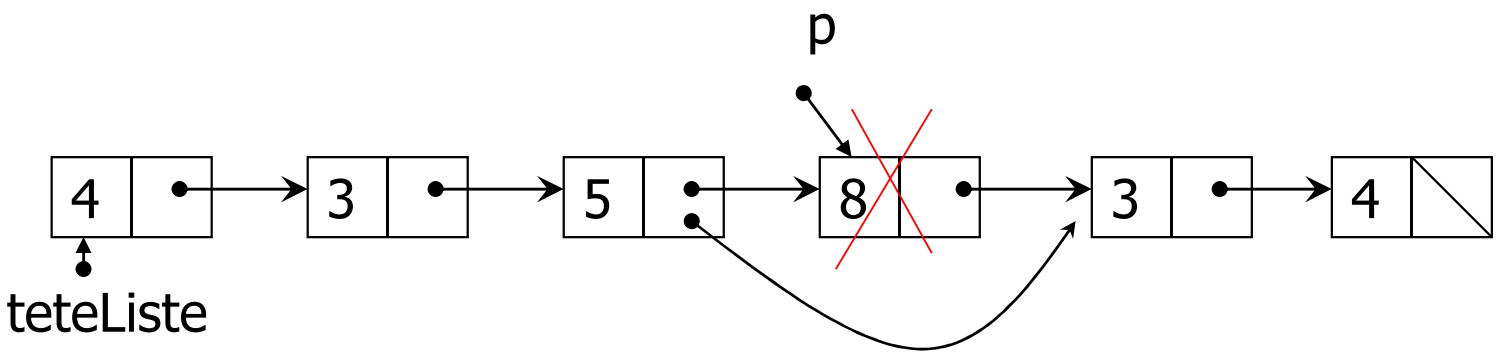
libérer(q)

Fin



Supprimer un nœud

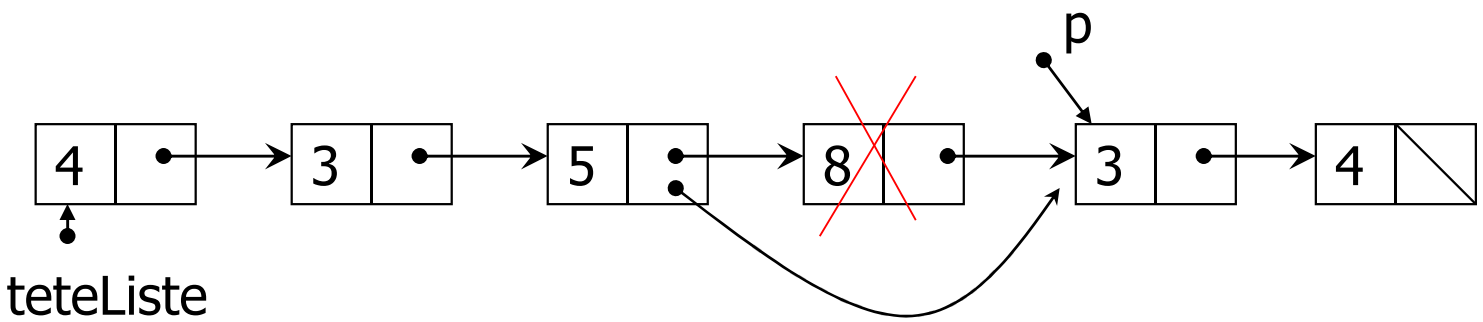
- Pour supprimer un nœud p, il faut trouver le prédécesseur de p pour établir le lien entre le prédécesseur de p et le successeur de p.



- ASTUCE : échanger les valeurs (de p et du successeur de p) puis supprimer l'élément après p

Supprimer avant un noeud

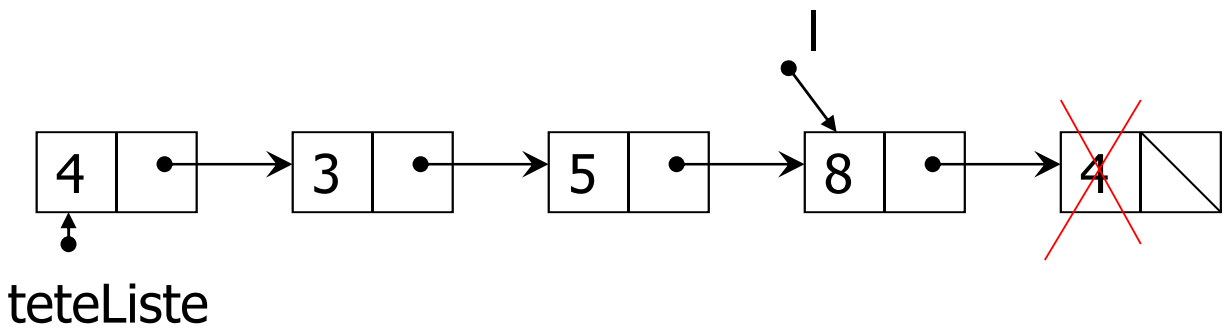
- Pour supprimer un élément avant p, il faut trouver le prédécesseur de prédécesseur de p pour établir le lien entre lui et p.



- Pas de solution, sinon de parcourir la liste dès le début pour trouver le prédécesseur du prédécesseur

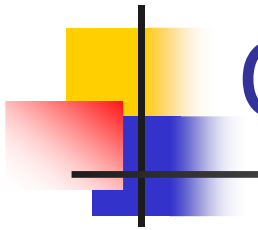
Supprimer en fin de liste

- Pour supprimer en fin de liste, il faut parcourir la liste pour trouver le dernier élément et puis le supprimer.



Supprimer en fin de liste

```
Fonction supprimerFin (teteListe : Pointeur): Pointeur
Variables l : Pointeur
Début
        Si (teteListe = null) Alors Retourner teteListe
        SinonSi (suivant(teteListe) = null)
                libérer(teteliste)
                Retourner null
        Sinon
                l ← teteListe
                Tantque (suivant(suivant(l)) ≠ null)
                        l ← suivant(l)
                Fintantque
                libérer(suivant(l))
                suivant(l) ← null
                Retourner teteListe
        FinSi
Fin
```



Comparaisons des implantations

Listes avec tableau	**Listes simplement chaînées**
Tout l'espace est alloué à l'avance	L'espace varie avec la liste
Pas d'espace autre que les valeurs	Chaque élément requiert de l'espace pour le(s) pointeur(s)
Insertion et suppression sont O(n)	Insertion et suppression sont O(1)
Précédent et accès direct sont O(1)	Précédent et accès direct sont O(n)