

# Algorithmique P2

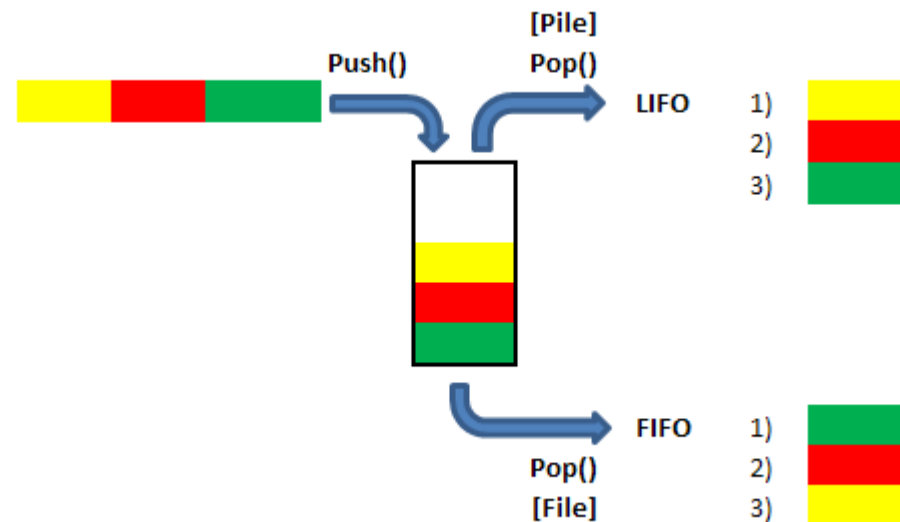
Piles, Files et Listes

Renaud Dumont, Ulg

2009–2010

# Piles et Files

- ▶ Dans une pile, l'élément supprimé est le dernier inséré
  - LIFO – Last In First Out – Dernier arrivé premier parti
- ▶ Dans une file, l'élément supprimé est le plus ancien
  - FIFO – First In First Out – Premier arrivé premier parti



# Piles

- ▶ Opérations
  - Empiler
  - Dépiler (Supprimer)
- ▶ Implémentation basée sur un tableau  $P$  de  $n$  éléments
  - Ordre déterminé par les indices des éléments du tableau  $P[1..n]$
  - Le tableau possède un attribut  $\text{sommet}[P]$  qui indexe l'élément le plus récemment inséré.
  - La pile se constitue des éléments  $P[1..\text{sommet}[P]]$  où  $P[1]$  est l'élément de base de la pile (le plus ancien) et  $P[\text{sommet}[P]]$  est l'élément situé au sommet (le plus récent).

# Piles

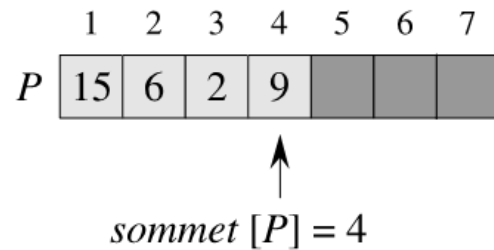
- Quand  $\text{sommet}[P] = 0$ , la pile est **vide**

PileVide (P)

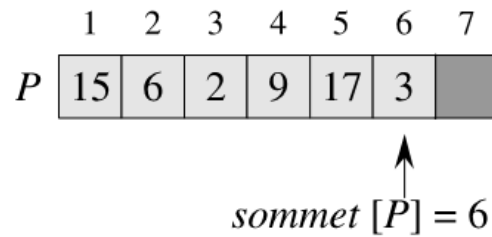
Si  $\text{sommet}[P] = 0$   
alors retourner Vrai  
sinon retourner Faux

- Si  $\text{sommet}[P]$  est supérieur à  $n$ , on dit que la pile **déborde**
  - Dépiler une pile vide provoque une erreur
    - On parle de **débordement négatif**
- Empiler (P,x)
  - $\text{Sommet}[P] \leftarrow \text{sommet}[P] + 1$
  - $P[\text{Sommet}[P]] \leftarrow x$
- Dépiler (P,x)
  - Si PileVide(P)
    - alors "débordement négatif" (Erreur)
  - sinon  $\text{sommet}[P] \leftarrow \text{sommet}[P] - 1$   
retourner  $P[\text{sommet}[P] + 1]$

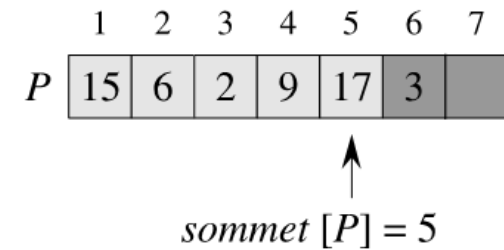
# Piles



(a)



(b)



(c)

## ▶ Remarques :

- En (c), la valeur 3 est conservée dans le tableau, bien que le sommet occupe la position 5.
- 3 n'est donc plus dans la pile, mais toujours présent dans le tableau.

# Files

- ▶ Opérations:
  - Enfiler
  - Défiler
- ▶ Une file possède une tête et une queue
  - Quand un élément est enfilé, il prend place en queue
  - Quand il est défilé, il est ôté de la tête de la file
- ▶ Implémentation à l'aide d'un tableau  $F$  de  $n$  éléments pour obtenir une file de  $n-1$  éléments
  - Ordre déterminé par les indices des éléments du tableau
  - $F[1..n]$

# Files

- La file a un attribut tête[F] qui repère sa tête
- La file a un attribut queue[F] qui indexe le prochain emplacement pouvant conserver un nouvel élément.
- Les éléments de la file sont repérés par tête[F], tête[F+1],... queue[F]-1
- L'emplacement 1 suit l'emplacement n dans un ordre circulaire
- Quand tête[F] = queue[F], la file est vide
  - Au départ, on a tête[F]=queue[F]=1
- Quand tête[F] = queue[F]+1, la file est pleine.

# Files

- ▶ Enfiler( $F, x$ )

$F[\text{queue}[F]] \leftarrow x$

Si  $\text{queue}[F] = \text{longueur}[F]$

alors  $\text{queue}[F] \leftarrow 1$

sinon  $\text{queue}[F] \leftarrow \text{queue}[F] + 1$

- ▶ Défiler( $F, x$ )

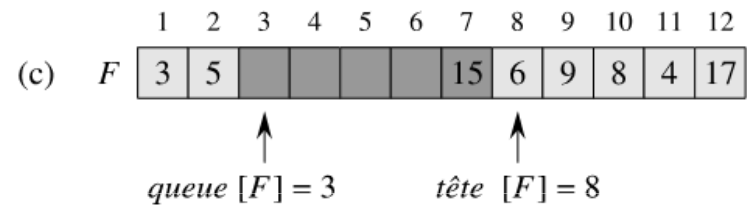
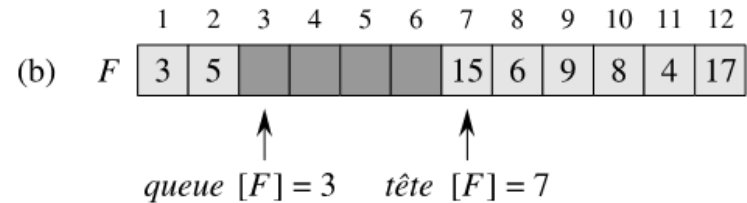
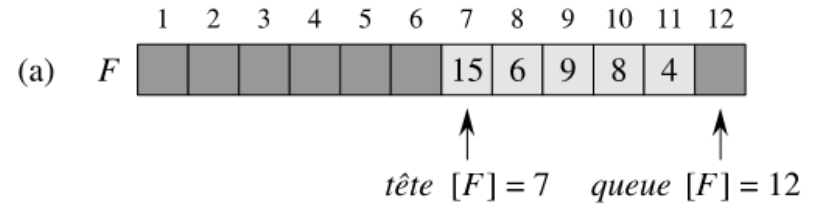
$x \leftarrow F[\text{tête}[F]]$

Si  $\text{tête}[F] = \text{longueur}[F]$

alors  $\text{tête}[F] \leftarrow 1$

sinon  $\text{tête}[F] \leftarrow \text{tête}[F] + 1$

retourner  $x$



# Listes chaînées

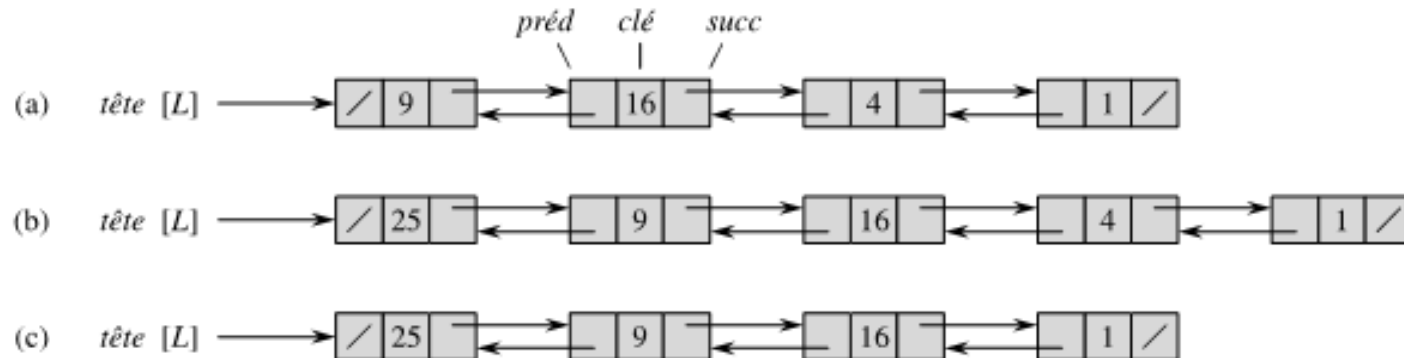
- ▶ Structure de données dans laquelle des objets de même type sont arrangés linéairement
- ▶ Ces objets constituent les éléments de la liste
- ▶ Ces éléments sont chaînés
  - L'ordre est déterminé par un pointeur dans chaque objet
- ▶ Chaque élément contient
  - Une partie de données
  - Un pointeur vers l'élément suivant (ou une marque de fin) → liste simplement chaînée
- ▶ Par la suite, on considère des listes doublement chaînées et non triées.

# Listes doublement chaînées

- ▶ Chaque élément d'une liste dispose de trois champs principaux
  - Un champ clé
    - Contient des données (éventuellement plusieurs champs)
  - Un champ succ : pointeur vers l'élément suivant
    - Pour un élément  $x$ ,  $\text{succ}[x]$  pointe le successeur
    - Si  $\text{succ}[x]=\text{NIL}$ , pas de successeur  $\rightarrow x$  est le dernier élément de la liste : la queue de liste
  - Un champ préd : pointeur vers l'élément précédent
    - Pour un élément  $x$ ,  $\text{préd}[x]$  pointe sur le prédécesseur
    - Si  $\text{préd}[x]=\text{NIL}$ , pas de prédécesseur  $\rightarrow x$  est le premier élément de la liste : la tête de liste

# Listes doublement chaînées

- Par souci d'efficacité de parcours, nous conservons deux pointeurs
  - tête : pointe sur le premier élément de la liste



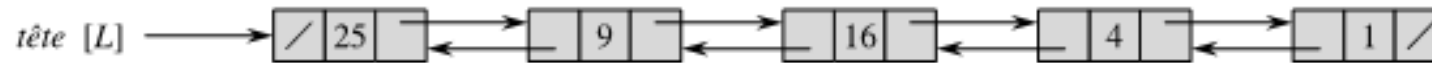
- queue : pointe vers le dernier élément de la liste.
  - En l'absence de ce derniers, il est toujours possible de parcourir la liste pour l'atteindre, mais coût ↗

# Types de listes

- ▶ Liste simplement chaînée
- ▶ Liste doublement chaînée
  - pointeur préd
- ▶ Liste triée
  - Les clés stockées sont triées
  - L'élément minimum est la tête de liste
  - L'élément maximum est la queue de liste
- ▶ Liste circulaire
  - Le pointeur préd de la tête de la liste pointe sur la queue de liste
  - Le pointeur succ de la queue de la liste pointe sur la tête de liste

# Listes – Opération : insertion

## ▶ Insérer-Liste(L,cur\_el)



### ◦ Insérer dans une liste vide

Préc[cur\_el] ← NIL

Succ[cur\_el] ← NIL

Tête[L] ← cur\_el

Queue[L] ← cur\_el

### ◦ Insérer un élément en tête de la liste

Préd[cur\_el] ← NIL

Succ[cur\_el] ← tête[L]

Préd[tête[L]] ← cur\_el

Tête[L] ← cur\_el

# Listes – Opération : insertion

- Insérer en fin de liste
  - Succ[cur\_el] ← NIL
  - Préd[cur\_el] ← queue[L]
  - Succ[Queue[L]] ← cur\_el
  - Queue ← cur\_el
- Insérer avant une position donnée (+ paramètre p)
  - p\_el ← tête[L]
  - Pour i ← 1 jusque p-1
    - p\_el ← succ[p\_el]
  - Succ[cur\_el] ← p\_el
  - Préd[cur\_el] ← Préd[p\_el]
  - Si Préd[p\_el] = NIL
    - Tête[L] ← cur\_el
  - Sinon Succ[Préd[p\_el]] ← cur\_el
  - Préd[p\_el] ← cur\_el

# Listes – Opération : insertion

- Insérer après une position donnée (+ paramètre p)

$p\_el \leftarrow \text{tête}[L]$

Pour  $i \leftarrow 1$  jusque  $p-1$

$p\_el \leftarrow \text{succ}[p\_el]$

$\text{Succ}[\text{cur\_el}] \leftarrow \text{Succ}[p\_el]$

$\text{Préd}[\text{cur\_el}] \leftarrow p\_el$

Si  $\text{Succ}[p\_el] = \text{NIL}$

$\text{Queue}[L] \leftarrow \text{cur\_el}$

Sinon  $\text{Préd}[\text{Succ}[p\_el]] \leftarrow \text{cur\_el}$

$\text{Succ}[p\_el] \leftarrow \text{cur\_el}$

# Listes – Opération : suppression

## ▶ Supprimer-Liste(L)

- Détruit un élément  $x$  de la liste  $L$  en mettant à jour les pointeurs des éléments successeur et précédent



- Suppression du premier élément

$Sup\_el \leftarrow tête[L]$

$Tête[L] \leftarrow succ[tête[L]]$

Si  $tête[L] = NIL$  // 1 seul élément

$queue[L] \leftarrow NIL$

Sinon

$préd[tête[L]] \leftarrow NIL$

$DEL(Sup\_el)$

# Listes – Opération : suppression

- Suppression du dernier élément
  - Sup\_el  $\leftarrow$  Queue[L]
  - Succ[Préd[Queue[L]]]  $\leftarrow$  NIL
  - Queue[L]  $\leftarrow$  Préd[Queue[L]]
  - DEL(Sup\_el)
- Suppression à une position donnée (Supprimer-Liste(L,p))
  - p\_el  $\leftarrow$  tête[L]
  - Pour i  $\leftarrow$  1 jusque p-1
    - p\_el  $\leftarrow$  succ[p\_el]
  - Sup\_el  $\leftarrow$  p\_el
  - Préd[succ[p\_el]]  $\leftarrow$  Préd[p\_el]
  - Succ[préd[p\_el]]  $\leftarrow$  Succ[p\_el]
  - DEL(Sup\_el)

# Listes – Opération : suppression

## ▶ Questions

- Pour les listes simplement liées, la suppression à une position donnée n'est pas possible. Il faudra supprimer l'élément *suivant* cette position.  
Pourquoi?
  - Pas de pointeur "préd", donc impossible de mettre à jour le pointeur "succ" de l'élément précédent.
- Que deviennent les éléments supprimés ?
  - Free (C – gestion dyn. de la mémoire)
  - Garbage collector (OO)

# Listes – Opérations

## ▶ En pratique

- Les listes disposent souvent d'un attribut "taille" en plus de "début" et "fin"
- Les insertions/suppressions tenant compte d'une position donnée comparent le paramètre p à cette taille pour déterminer le type d'insertion/suppression à effectuer

Insérer-Liste(L,cur\_el,p)

Supprimer-Liste(L, p)

- En paramètre d'une fonction, on passera

- Un pointeur vers cur\_el, ou

Insérer-Liste(L,cur\_el,p)

- Une donnée et l'on créera l'élément dans le corps de ladite fonction

Insérer-Liste(L,val,p)

"Créer un élément de la liste de champs de données égal à val"

...

# Listes – Opération : recherche

- ▶ Recherche-Liste(L,k)
  - Trouve le premier élément dont la clé est k dans la liste L
    - Retourne un pointeur sur cet élément
    - Retourne NIL sinon



$p\_el \leftarrow tête[L]$

Tant que  $p\_el \neq NIL$  et  $clé[p\_el] \neq k$

$p\_el \leftarrow succ[p\_el]$

Retourner  $p\_el$

# Retour aux piles et files

- ▶ Piles et files avec liste chaînée
  - Pile = liste simplement chaînée dont on ne traite que l'élément en tête de liste (= le premier élément).
  - File = liste simplement chaînée dont on traite les extrémités.

# Retour aux piles et files

## ▶ Pile

- Insertion dans une pile vide  
 $\text{succ}[\text{cur\_el}] \leftarrow \text{tête}[L]$   
 $\text{tête}[L] \leftarrow \text{cur\_el}$
- Insertion dans une pile  
 $\text{succ}[\text{cur\_el}] \leftarrow \text{tête}[L]$   
 $\text{tête}[L] \leftarrow \text{cur\_el}$
- Suppression d'un élément  
 $\text{sup\_el} \leftarrow \text{tête}[L]$   
 $\text{tête}[L] \leftarrow \text{succ}[\text{tête}[L]]$   
 $\text{DEL}(\text{sup\_el})$

# Retour aux piles et files

## ▶ File

- Insertion dans une file vide
  - $\text{succ}[\text{cur\_el}] \leftarrow \text{NIL}$
  - $\text{queue}[L] \leftarrow \text{cur\_el}$
  - $\text{tête}[L] \leftarrow \text{cur\_el}$
- Insertion dans une file
  - $\text{succ}[\text{cur\_el}] \leftarrow \text{NIL}$
  - $\text{succ}[\text{queue}[L]] \leftarrow \text{cur\_el}$
  - $\text{queue}[L] \leftarrow \text{cur\_el}$
- Suppression d'un élément
  - $\text{sup\_el} \leftarrow \text{tête}[L]$
  - $\text{tête}[L] \leftarrow \text{succ}[\text{tête}[L]]$
  - $\text{DEL}(\text{sup\_el})$

# Listes liées : exercices

- ▶ Soit une liste L. Supprimer un élément sur deux (en gardant celui de tête)
- ▶ Soit une liste L. Si la clé d'un élément p est paire (soit  $\text{clé}[p] \bmod 2 = 0$ ), la remplacer par deux éléments dont les clés sont égales à  $\text{clé}[p]/2$ .
- ▶ Soit une liste L. On veut retourner la liste.

# Listes liées : exercices

- ▶ Soit une liste L. Supprimer un élément sur deux (en gardant celui de tête)

$p \leftarrow \text{tête}[L]$

Tant que  $p \neq \text{nil}$  et  $\text{succ}[p] \neq \text{NIL}$

$q \leftarrow \text{succ}[p]$

$\text{succ}[p] \leftarrow \text{succ}[q]$

$\text{DEL}(q)$

$p \leftarrow \text{succ}[p]$

# Listes liées : exercices

- ▶ Soit une liste L. Si la clé d'un élément p est paire (soit  $\text{clé}[p] \bmod 2 = 0$ ), la remplacer par deux éléments dont les clés sont égales à  $\text{clé}[p]/2$ .

```
p ← tête[L]
Tant que p ≠ NIL
  si (clé[p] mod 2 = 1)
    alors p ← succ[p]
  sinon NEW(q)
    clé[q] ← clé[p]/2
    clé[p] ← clé[p]/2
    succ[q] ← succ[p]
    succ[p] ← q
    p ← succ[q]
```

# Listes liées : exercices

- ▶ Soit une liste L. On veut retourner la liste.

```
p ← tête[L]
tête[L] ← nil
Tant que p ≠ nil
  q ← Succ[p]
  succ[p] ← tête[L]
  tête[L] ← p
  p ← q
```