

# C8 Structures de données linéaires

## 1. Généralités

### Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

### Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Les tableaux fixes du C sont un exemple de structure de données.

### Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Les tableaux fixes du C sont un exemple de structure de données.

- L'**interface** de la structure de données est l'ensemble des opérations accessibles à un utilisateur de la structure de données.

### Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Les tableaux fixes du C sont un exemple de structure de données.

- L'**interface** de la structure de données est l'ensemble des opérations accessibles à un utilisateur de la structure de données.

Par exemple, la notation `[]` permet d'accéder à un élément du tableau, pour le lire ou le modifier. Par contre, la taille du tableau ne fait pas partie de l'interface.

### Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Les tableaux fixes du C sont un exemple de structure de données.

- L'**interface** de la structure de données est l'ensemble des opérations accessibles à un utilisateur de la structure de données.

Par exemple, la notation `[]` permet d'accéder à un élément du tableau, pour le lire ou le modifier. Par contre, la taille du tableau ne fait pas partie de l'interface.

- L'**implémentation** de la structure de données est la façon dont elle est représentée et codée en mémoire et n'est pas forcément accessible à l'utilisateur.

### Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Les tableaux fixes du C sont un exemple de structure de données.

- L'**interface** de la structure de données est l'ensemble des opérations accessibles à un utilisateur de la structure de données.

Par exemple, la notation `[]` permet d'accéder à un élément du tableau, pour le lire ou le modifier. Par contre, la taille du tableau ne fait pas partie de l'interface.

- L'**implémentation** de la structure de données est la façon dont elle est représentée et codée en mémoire et n'est pas forcément accessible à l'utilisateur.

On peut utiliser les listes de OCaml via leur interface sans savoir comment elles sont représentées en mémoire par le langage.

### Caractérisation par l'interface

- La différence entre **interface** et **implémentation** est fondamentale et doit être bien comprise. En effet une même structure de données peut avoir plusieurs implémentations. L'idée est que l'utilisation de la structure de données doit se faire indépendamment de son implémentation ce qui permet la séparation des programmes en composants indépendants (modularité).

On utilise la même interface (les opérations arithmétiques) pour manipuler les entiers du C et de Python mais ils ne sont pas implémentés de la même manière.

### Caractérisation par l'interface

- La différence entre **interface** et **implémentation** est fondamentale et doit être bien comprise. En effet une même structure de données peut avoir plusieurs implémentations. L'idée est que l'utilisation de la structure de données doit se faire indépendamment de son implémentation ce qui permet la séparation des programmes en composants indépendants (modularité).  
On utilise la même interface (les opérations arithmétiques) pour manipuler les entiers du C et de Python mais ils ne sont pas implémentés de la même manière.
- Lorsqu'on définit un *cahier des charges* pour une structure de données (ensemble des données et opérations possibles), on définit ce qu'on appelle un **type abstrait de données**. Ainsi, une structure de données peut être vue comme une implémentation d'un type abstrait de données.



# C8 Structures de données linéaires

## 1. Généralités

### Caractérisation par l'interface

- La différence entre **interface** et **implémentation** est fondamentale et doit être bien comprise. En effet une même structure de données peut avoir plusieurs implémentations. L'idée est que l'utilisation de la structure de données doit se faire indépendamment de son implémentation ce qui permet la séparation des programmes en composants indépendants (modularité).

On utilise la même interface (les opérations arithmétiques) pour manipuler les entiers du C et de Python mais ils ne sont pas implémentés de la même manière.

- Lorsqu'on définit un *cahier des charges* pour une structure de données (ensemble des données et opérations possibles), on définit ce qu'on appelle un **type abstrait de données**. Ainsi, une structure de données peut être vue comme une implémentation d'un type abstrait de données.
- La définition complète d'un type abstrait de données inclut généralement la complexité des opérations de l'interface.

L'ajout d'un élément en tête d'une liste de OCaml est une opération en  $O(1)$ .

### Opérations de l'interface

- La création d'une structure de données se fait à l'aide d'une opération de l'interface appelé **constructeur**


### Opérations de l'interface

- La création d'une structure de données se fait à l'aide d'une opération de l'interface appelé **constructeur**  
Par exemple en C, **double** tab[10] ;
- La récupération d'une valeur dans la structure se fait à l'aide d'**accesseur** (en anglais *getter*).

### Opérations de l'interface

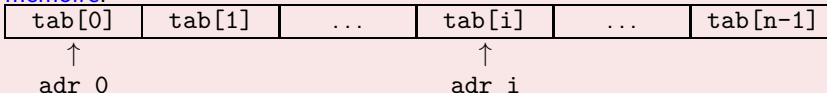
- La création d'une structure de données se fait à l'aide d'une opération de l'interface appelé **constructeur**  
Par exemple en C, **double** tab[10];
- La récupération d'une valeur dans la structure se fait à l'aide d'**accesseur** (en anglais *getter*).  
Par exemple en C, **double** e = tab[3];
- La modification d'une valeur dans la structure se fait à l'aide de **transformateur** (en anglais *setter*).  
Par exemple en C, tab[3] = 7.5;

### Opérations de l'interface

- La création d'une structure de données se fait à l'aide d'une opération de l'interface appelé **constructeur**  
Par exemple en C, **double** `tab[10]` ;
- La récupération d'une valeur dans la structure se fait à l'aide d'**accesseur** (en anglais *getter*).  
Par exemple en C, **double** `e = tab[3]` ;
- La modification d'une valeur dans la structure se fait à l'aide de **transformateur** (en anglais *setter*).  
Par exemple en C, `tab[3] = 7.5` ;  
 On distingue les structures de données mutables (comme les tableaux du C) des structures de données immuables (comme les listes de OCaml). En cas de non mutabilité, pour modifier une structure de données on doit en construire une nouvelle.

### Définition

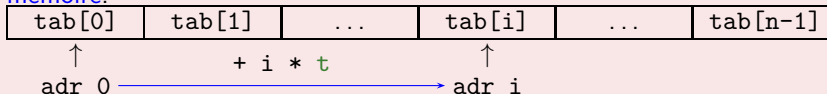
- un tableau est une séquence de  $n$  valeurs de même type consécutives en mémoire.





## Définition

- un tableau est une séquence de  $n$  valeurs de même type **consécutives en mémoire**.

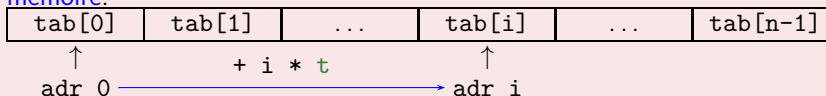


- L'accès à un élément se fait **temps constant**, en effet il suffit de connaître la taille  $t$  d'une case et de disposer de l'adresse du premier élément du tableau  $\text{adr0}$ . L'adresse de l'élément d'indice  $i$  s'obtient alors en ajoutant à l'adresse du premier élément  $i$  fois la taille d'une case.



### Définition

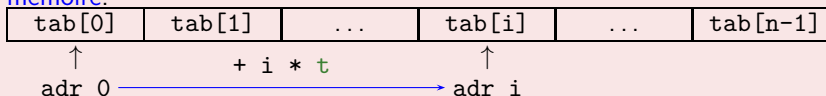
- un tableau est une séquence de  $n$  valeurs de même type **consécutives en mémoire**.



- L'accès à un élément se fait **temps constant**, en effet il suffit de connaître la taille  $t$  d'une case et de disposer de l'adresse du premier élément du tableau  $adr0$ . L'adresse de l'élément d'indice  $i$  s'obtient alors en ajoutant à l'adresse du premier élément  $i$  fois la taille d'une case.
- La suppression ou l'insertion d'un élément demande par contre la recopie des éléments et ce sont donc des opérations en  $O(n)$ .

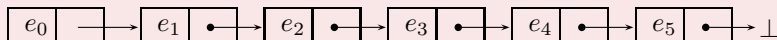
### Définition

- un tableau est une séquence de  $n$  valeurs de même type **consécutives en mémoire**.



- L'accès à un élément se fait **temps constant**, en effet il suffit de connaître la taille  $t$  d'une case et de disposer de l'adresse du premier élément du tableau  $adr_0$ . L'adresse de l'élément d'indice  $i$  s'obtient alors en ajoutant à l'adresse du premier élément  $i$  fois la taille d'une case.
- La suppression ou l'insertion d'un élément demande par contre la recopie des éléments et ce sont donc des opérations en  $O(n)$ .
- Les tableaux de OCaml seront vus au chapitre suivant, on se limite donc pour le moment à l'utilisation de ceux du C.

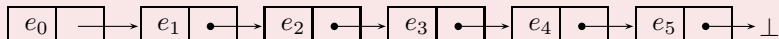
### Définition



# C8 Structures de données linéaires

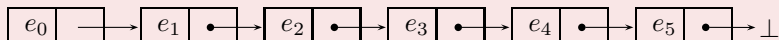
## 3. Liste chaînées

### Définition



- Contrairement aux tableaux, les différentes valeurs ne sont pas stockées de façon contiguës en mémoire.

### Définition

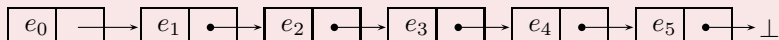


- Contrairement aux tableaux, les différentes valeurs ne sont pas stockées de façon contiguës en mémoire.
- Avec chaque élément, on stocke aussi dans un « maillon » l'emplacement de son successeur. Le successeur du dernier élément se note  $\perp$ .

# C8 Structures de données linéaires

## 3. Liste chaînées

### Définition

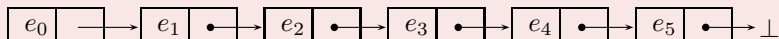


- Contrairement aux tableaux, les différentes valeurs ne sont pas stockées de façon contiguës en mémoire.
- Avec chaque élément, on stocke aussi dans un « maillon » l'emplacement de son successeur. Le successeur du dernier élément se note  $\perp$ .
- Pour accéder à un élément, on doit parcourir tout ceux qui le précèdent. L'accès à un élément est donc une opération en  $O(n)$ .

# C8 Structures de données linéaires

## 3. Liste chaînées

### Définition

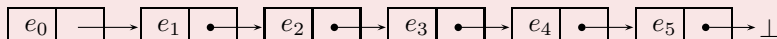


- Contrairement aux tableaux, les différentes valeurs ne sont pas stockées de façon contiguës en mémoire.
- Avec chaque élément, on stocke aussi dans un « maillon » l'emplacement de son successeur. Le successeur du dernier élément se note  $\perp$ .
- Pour accéder à un élément, on doit parcourir tout ceux qui le précèdent. L'accès à un élément est donc une opération en  $O(n)$ .
- L'ajout ou la suppression en tête de liste est en  $O(1)$ , la taille de la structure de données n'est pas fixé à la construction contrairement aux tableaux.

# C8 Structures de données linéaires

## 3. Liste chaînées

### Définition



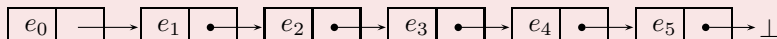
- Contrairement aux tableaux, les différentes valeurs ne sont pas stockées de façon contiguës en mémoire.
- Avec chaque élément, on stocke aussi dans un « maillon » l'emplacement de son successeur. Le successeur du dernier élément se note  $\perp$ .
- Pour accéder à un élément, on doit parcourir tout ceux qui le précèdent. L'accès à un élément est donc une opération en  $O(n)$ .
- L'ajout ou la suppression en tête de liste est en  $O(1)$ , la taille de la structure de données n'est pas fixée à la construction contrairement aux tableaux.
- Les listes chaînées peuvent être définies de façon **récursive** :



## C8 Structures de données linéaires

### 3. Liste chaînées

#### Définition

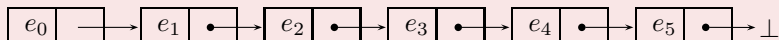


- Contrairement aux tableaux, les différentes valeurs ne sont pas stockées de façon contiguës en mémoire.
- Avec chaque élément, on stocke aussi dans un « maillon » l'emplacement de son successeur. Le successeur du dernier élément se note  $\perp$ .
- Pour accéder à un élément, on doit parcourir tout ceux qui le précèdent. L'accès à un élément est donc une opération en  $O(n)$ .
- L'ajout ou la suppression en tête de liste est en  $O(1)$ , la taille de la structure de données n'est pas fixée à la construction contrairement aux tableaux.
- Les listes chaînées peuvent être définies de façon **récursive** :
  - Une liste est soit vide (référence vers  $\perp$ )

# C8 Structures de données linéaires

## 3. Liste chaînées

### Définition



- Contrairement aux tableaux, les différentes valeurs ne sont pas stockées de façon contiguës en mémoire.
- Avec chaque élément, on stocke aussi dans un « maillon » l'emplacement de son successeur. Le successeur du dernier élément se note  $\perp$ .
- Pour accéder à un élément, on doit parcourir tout ceux qui le précèdent. L'accès à un élément est donc une opération en  $O(n)$ .
- L'ajout ou la suppression en tête de liste est en  $O(1)$ , la taille de la structure de données n'est pas fixée à la construction contrairement aux tableaux.
- Les listes chaînées peuvent être définies de façon **récursive** :
  - Une liste est soit vide (référence vers  $\perp$ )
  - Soit c'est la donnée d'un maillon constitué d'une valeur et d'une référence vers une liste.

# C8 Structures de données linéaires

## 3. Liste chaînées

### Implémentation en C

On peut définir un maillon comme un **struct** avec les champs valeur et pointeur vers un maillon :

```
1 struct maillon
2 {
3     int valeur;
4     struct maillon * suivant;
5 };
6 typedef struct maillon maillon;
```

### Implémentation en OCaml

- Le type 'a list est prédéfini dans le langage
- Attention, les listes de OCaml ne sont pas mutables
- Tous les éléments doivent être du même type

## Piles

- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.

## Piles

- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.

elt4
elt3
elt2
elt1

## Piles

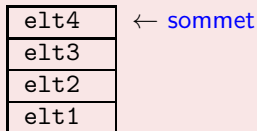
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.

elt4
elt3
elt2
elt1

- L'élément situé en haut de la pile s'appelle le **sommet**.

## Piles

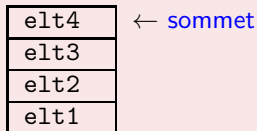
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.

## Piles

- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.

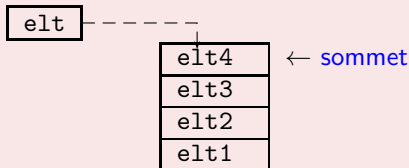


- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile



## Piles

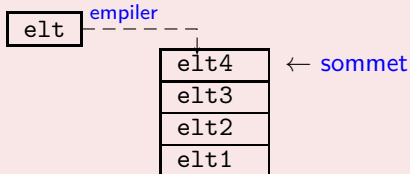
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile

## Piles

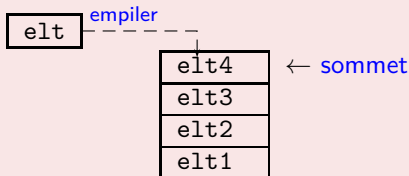
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile

## Piles

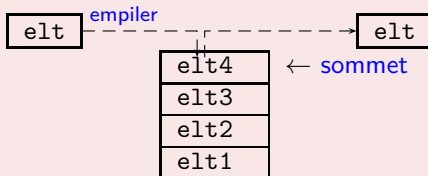
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile
- Dépiler signifie retirer l'élément situé au sommet de la pile

## Piles

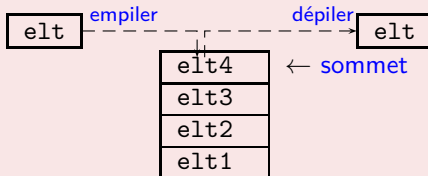
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile
- Dépiler signifie retirer l'élément situé au sommet de la pile

## Piles

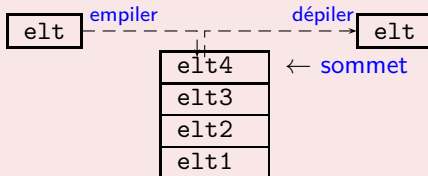
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile
- Dépiler signifie retirer l'élément situé au sommet de la pile

## Piles

- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile
- Dépiler signifie retirer l'élément situé au sommet de la pile
- Ainsi le premier élément entré dans la pile sera aussi le dernier à en sortir, on dit qu'une pile est une structure **LIFO** *Last In First Out*

## Piles comme structures de données

L'interface d'une structure de données Pile se limite donc aux opérations suivantes :

## Piles comme structures de données

L'interface d'une structure de données Pile se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la pile est vide ou non.



## Piles comme structures de données

L'interface d'une structure de données Pile se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la pile est vide ou non.
- `empiler()` (en anglais *push*) qui ajoute un élément au sommet de la pile.

## Piles comme structures de données

L'interface d'une structure de données Pile se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la pile est vide ou non.
- `empiler()` (en anglais *push*) qui ajoute un élément au sommet de la pile.
- `depiler()` (en anglais *pop*) qui retire l'élément situé au sommet (cela n'est possible que si la pile n'est pas vide).

## Piles comme structures de données

L'interface d'une structure de données Pile se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la pile est vide ou non.
- `empiler()` (en anglais *push*) qui ajoute un élément au sommet de la pile.
- `depiler()` (en anglais *pop*) qui retire l'élément situé au sommet (cela n'est possible que si la pile n'est pas vide).

## Utilisation

En dépit de sa simplicité, cette structure de données a de nombreuses applications en informatique : pile d'appel récursif, pile d'évaluation d'une expression, ...

## Manipulation de piles

## Manipulation de piles

- On considère la pile :  $P = | "A", "L", "I", "X" >$  (le sommet est "X"). Quelle suite d'opération permet d'obtenir  $P = | "A", "L", "E", "X" >$ ?

## Manipulation de piles

- On considère la pile :  $P = | "A", "L", "I", "X" >$  (le sommet est "X"). Quelle suite d'opération permet d'obtenir  $P = | "A", "L", "E", "X" >$ ?
- Un programmeur décide d'utiliser une pile afin de stocker une réponse entrée au clavier. Chaque caractère tapé doit être empiler. Traduire en terme d'opérations sur cette pile les actions suivantes :

## Manipulation de piles


- On considère la pile :  $P = | "A", "L", "I", "X" >$  (le sommet est "X"). Quelle suite d'opération permet d'obtenir  $P = | "A", "L", "E", "X" >$ ?
- Un programmeur décide d'utiliser une pile afin de stocker une réponse entrée au clavier. Chaque caractère tapé doit être empiler. Traduire en terme d'opérations sur cette pile les actions suivantes :
  - Appuie sur la touche "o"

## Manipulation de piles


- On considère la pile :  $P = | "A", "L", "I", "X" >$  (le sommet est "X"). Quelle suite d'opération permet d'obtenir  $P = | "A", "L", "E", "X" >$ ?
- Un programmeur décide d'utiliser une pile afin de stocker une réponse entrée au clavier. Chaque caractère tapé doit être empiler. Traduire en terme d'opérations sur cette pile les actions suivantes :
  - Appuie sur la touche "o"
  - Appuie sur la touche "i"



## Manipulation de piles

- On considère la pile :  $P = | "A", "L", "I", "X" >$  (le sommet est "X"). Quelle suite d'opération permet d'obtenir  $P = | "A", "L", "E", "X" >$ ?
- Un programmeur décide d'utiliser une pile afin de stocker une réponse entrée au clavier. Chaque caractère tapé doit être empiler. Traduire en terme d'opérations sur cette pile les actions suivantes :
  - Appuie sur la touche "o"
  - Appuie sur la touche "i"
  - Appuie sur la touche  (*backspace*)

## Manipulation de piles

- On considère la pile :  $P = | "A", "L", "I", "X" >$  (le sommet est "X"). Quelle suite d'opération permet d'obtenir  $P = | "A", "L", "E", "X" >$ ?
- Un programmeur décide d'utiliser une pile afin de stocker une réponse entrée au clavier. Chaque caractère tapé doit être empiler. Traduire en terme d'opérations sur cette pile les actions suivantes :
  - Appuie sur la touche "o"
  - Appuie sur la touche "i"
  - Appuie sur la touche  (*backspace*)
  - Appuie sur la touche "k"

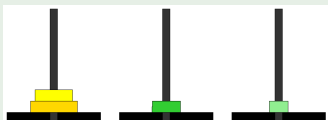
## Manipulation de piles

Au jeu des tours des Hanoï, on gère les trois tours T1, T2 et T3 à l'aide de trois piles.

## Manipulation de piles

Au jeu des tours des Hanoï, on gère les trois tours T1, T2 et T3 à l'aide de trois piles.

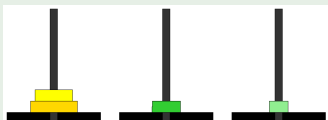
- 1 Quel est le contenu de chacune des piles dans la situation ci-dessous? (un disque est représenté dans la pile par sa taille)



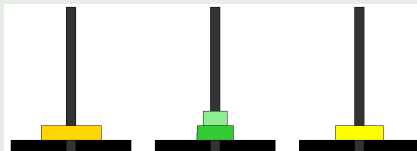
## Manipulation de piles

Au jeu des tours des Hanoï, on gère les trois tours T1, T2 et T3 à l'aide de trois piles.

- 1 Quel est le contenu de chacune des piles dans la situation ci-dessous ? (un disque est représenté dans la pile par sa taille)



- 2 Ecrire les opérations permettant de passer la situation précédente à celle ci-dessous :



## Implémentation des piles

Plusieurs implémentations sont possibles :

- A l'aide d'une liste chaînée, le début de la liste représente alors le sommet.

## Implémentation des piles

Plusieurs implémentations sont possibles :

- A l'aide d'une liste chaînée, le début de la liste représente alors le sommet.
- Si la capacité de la pile est bornée et connue en amont, on peut aussi utiliser un tableau et mémoriser la taille  $t$  de la pile. Puis,

## Implémentation des piles

Plusieurs implémentations sont possibles :

- A l'aide d'une liste chaînée, le début de la liste représente alors le sommet.
- Si la capacité de la pile est bornée et connue en amont, on peut aussi utiliser un tableau et mémoriser la taille  $t$  de la pile. Puis,
  - pour tester si la pile est vide on teste si  $t$  est égal à 0,



## Implémentation des piles

Plusieurs implémentations sont possibles :

- A l'aide d'une liste chaînée, le début de la liste représente alors le sommet.
- Si la capacité de la pile est bornée et connue en amont, on peut aussi utiliser un tableau et mémoriser la taille  $t$  de la pile. Puis,
  - pour tester si la pile est vide on teste si  $t$  est égal à 0,
  - pour empiler une valeur  $v$ , on affecte  $\text{tab}[t]=v$  et on incrémente  $t$ ,

## Implémentation des piles

Plusieurs implémentations sont possibles :

- A l'aide d'une liste chaînée, le début de la liste représente alors le sommet.
- Si la capacité de la pile est bornée et connue en amont, on peut aussi utiliser un tableau et mémoriser la taille  $t$  de la pile. Puis,
  - pour tester si la pile est vide on teste si  $t$  est égal à 0,
  - pour empiler une valeur  $v$ , on affecte  $\text{tab}[t]=v$  et on incrémente  $t$ ,
  - pour dépiler renvoie  $\text{tab}[t]$  et décrémente  $t$ .

## Files

- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.

## Files

- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.

elt4	elt3	elt2	elt1
------	------	------	------

## Files

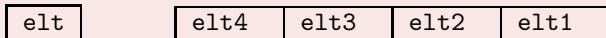
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file

## Files

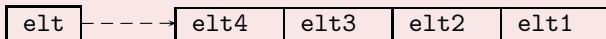
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file

## Files

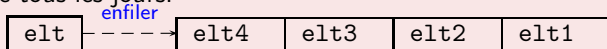
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file

## Files

- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.

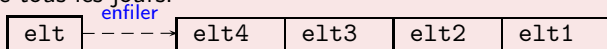


- Enfiler signifie ajouter un élément en fin de file



## Files

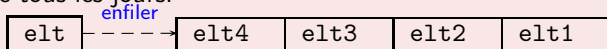
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file
- Défiler signifie retirer l'élément situé au début de la file.

## Files

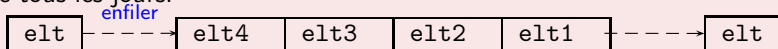
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file
- Défiler signifie retirer l'élément situé au début de la file.

## Files

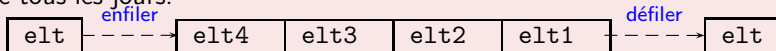
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file
- Défiler signifie retirer l'élément situé au début de la file.

## Files

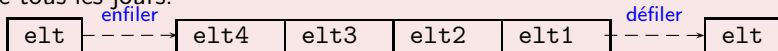
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file
- Défiler signifie retirer l'élément situé au début de la file.

## Files

- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file
- Défiler signifie retirer l'élément situé au début de la file.
- Ainsi le premier élément entré dans la file sera aussi le premier à en sortir, on dit qu'une file est une structure **FIFO** *First In First Out*

## Files comme structures de données

L'interface d'une structure de données File se limite donc aux opérations suivantes :

## Files comme structures de données

L'interface d'une structure de données File se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la file est vide ou non.

## Files comme structures de données

L'interface d'une structure de données File se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la file est vide ou non.
- `enfiler(element)` qui ajoute un élément à la fin de la file.



## Files comme structures de données

L'interface d'une structure de données File se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la file est vide ou non.
- `enfiler(element)` qui ajoute un élément à la fin de la file.
- `defiler()` qui retire l'élément situé au début de la file (cela n'est possible que si la file n'est pas vide).

## Files comme structures de données

L'interface d'une structure de données File se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la file est vide ou non.
- `enfiler(element)` qui ajoute un élément à la fin de la file.
- `defiler()` qui retire l'élément situé au début de la file (cela n'est possible que si la file n'est pas vide).

## Utilisation

Comme pour les piles, cette structure de données a de nombreuses applications en informatique : file d'attente d'une imprimante, simulation de files d'attentes réelles, ...

## Manipulation de files

## Manipulation de files

- On considère la file :  $F = \langle "E", "L", "S", "A" \rangle$ . Quelle suite d'opération permet d'obtenir  $F = \langle "N", "O", "E", "L" \rangle$ ?

## Manipulation de files

- On considère la file :  $F = \langle "E", "L", "S", "A" \rangle$ . Quelle suite d'opération permet d'obtenir  $F = \langle "N", "O", "E", "L" \rangle$ ?
- On simule la file d'attente d'une imprimante à l'aide d'une file. A quelle opération sur cette file correspond l'envoi d'une nouvelle impression ? La fin de l'impression en cours ? Dans quel ordre seront effectuées les impressions ?

## Implémentation des files

Plusieurs implémentations sont possibles :

- A l'aide d'une liste chaînée, dans ce cas si on veut pouvoir enfiler et défiler en temps constant, il faut disposer d'un accès au premier maillon et au dernier maillon.

## Implémentation des files

Plusieurs implémentations sont possibles :

- A l'aide d'une liste chaînée, dans ce cas si on veut pouvoir enfiler et défiler en temps constant, il faut disposer d'un accès au premier maillon et au dernier maillon.
- Si la capacité de la file est bornée et connue en amont, on peut aussi utiliser un tableau de façon circulaire en mémorisant l'indice du début de la file et l'indice de fin.

## Implémentation des files

Plusieurs implémentations sont possibles :

- A l'aide d'une liste chaînée, dans ce cas si on veut pouvoir enfiler et défiler en temps constant, il faut disposer d'un accès au premier maillon et au dernier maillon.
- Si la capacité de la file est bornée et connue en amont, on peut aussi utiliser un tableau de façon circulaire en mémorisant l'indice du début de la file et l'indice de fin.
- On peut aussi implémenter une file avec deux piles.