UNIVERSITE ABDELMALEK ESSAADI

Faculté des Sciences de Tétouan

CHAPITRE 2

Les Listes chainées, les piles et les files

a.souri@uae.ac.ma
adnan.souri@gmail.com

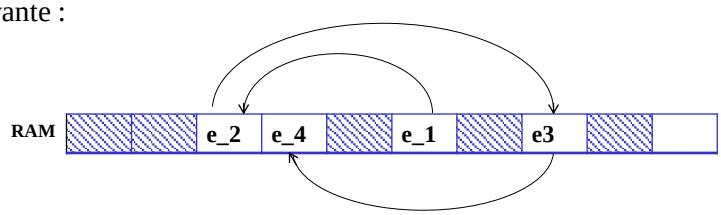
Les listes chainées

- Un tableau de taille 4 ne peut plus occuper d'espace sur cette RAM !

- Une liste chainée est un formalisme informatique qui permet de sauvegarder les données de manière non contiguë et dynamique.
- Allocation plus transparente
- On n'est pas obligé de savoir à priori le nombre de données

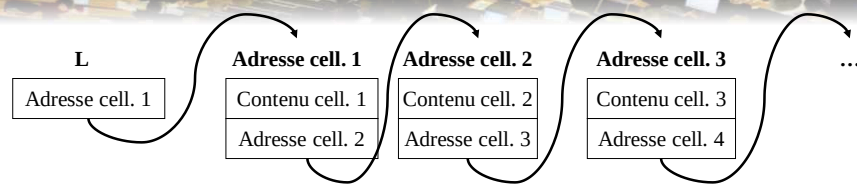
Les listes chainées

- Les éléments d'une liste chainée peuvent être illustrés de la manière suivante :



22

Les listes chainées



- La dernière cellule de la liste pointe sur l'adresse NULL
- En mémoire :

	@c3			@c1			L		@c2		
RAM	…	c3	@c4	…	c1	@c2	…	@c1	…	c2	@c3	…

23

Les listes chainées

- **Structures auto-référentes**
- Principe utilisé pour les listes chaînées :

```
struct cellule {
  ... // champs correspondant au contenu
  struct cellule *suivant ;
} ;
```

- La tête de la liste est un pointeur sur la première cellule

Les listes chainées

		@c3			@c1			L		@c2		
RAM	…	c3	@c4	…	c1	@c2	…	@c1	…	c2	@c3	…

- Chaque élément d'une liste chaînée est composé de deux parties :
  - Le contenu (la valeur) qu'on veut stocker
  - L'adresse de l'élément suivant (s'il existe).
- S'il n'y a plus d'élément suivant, alors l'adresse sera NULL, et désignera la fin de la liste chaînée.

Les listes chainées

- Maintenant que nous savons comment déclarer une liste chaînée, il serait intéressant d'apprendre à réaliser des opérations indispensables au traitement des listes (parcours, ajout d'éléments, modification, suppression)
- C'est ce que nous allons étudier dans cette partie sur la manipulation des listes chaînées.
- Dans tous les cas, on renvoie la nouvelle liste, c'est-à-dire un pointeur sur élément contenant l'adresse du premier élément de la liste.

Les listes chainées

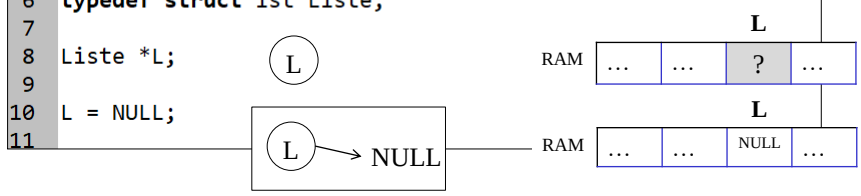
- **Manipulations usuelles :**
  - Créer une liste, éventuellement vide
  - Parcourir une liste
  - Calculer le nombre d'éléments (la taille d'une liste)
  - Ajouter un élément à une liste (début, fin, milieu)
  - Supprimer un élément d'une liste (début, fin, milieu)
  - Trier les éléments d'une liste
  - Détruire une liste
  - …

Voir les codes sources en C

Les listes chainées

- Création d'une liste vide

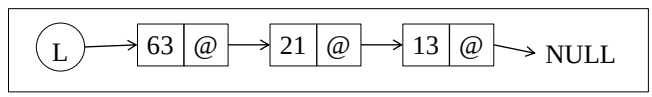
```
1  struct lst {
2      int x;
3      struct lst *suiv;
4  };                          // Définition de la structure lst
5
6  typedef struct lst Liste;
7
8  Liste *L;
9
10 L = NULL;
11
```



Les listes chainées

- Parcourir une liste
- Supposons qu'on veut afficher tous les éléments d'une liste. Il faut la parcourir et afficher un par un.

```
15
16 while(L!=NULL){
17         printf("%d ",L->contenu);
18         L = L->suivant;
19     }
```



Allocation dynamique

- Ajouter un élément à une liste
- A chaque fois que l'on veut ajouter un nouvel élément, on alloue l'espace pour cet élément (***allocation dynamique* – fonction *malloc***)

- On ne savait pas *a priori* combien de cellules vont constituer la liste

  ⇒ impossible de réserver l'espace mémoire nécessaire au début du programme (comme avec la représentation par tableaux – *allocation statique*)
  ⇒ il faut réserver l'espace nécessaire *pendant l'exécution*

  Intérêt : seul l'espace réellement nécessaire est réservé

30

Les listes chainées

- Ajouter un élément à une liste : **Principe général**

1. Allocation dynamique d'espace mémoire (Pointeur qui pointe vers cet espace nouvellement alloué)
2. Remplir les champs du nouvel élément *x* (le contenu)
3. Préciser **ou** insérer le nouvel élément *x* dans la liste **(début, milieu, fin)**

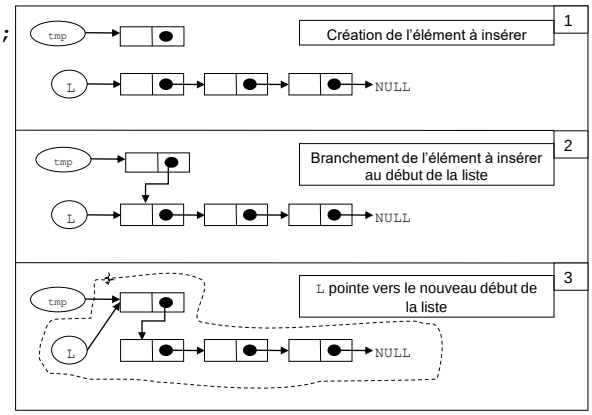
4. Remplir le champ suivant de *x*
5. L'élément qui va être avant x dans la liste doit changer son pointeur suivant. Il doit pointer vers x.
6. Incrémenter la variable qui stocke la taille de liste

31

Les listes chainées

- Ajouter un élément au début d'une liste

```
tmp =(struct Liste*)malloc(sizeof(struct Liste));
printf("ajouter :");
scanf("%d",&v);
tmp->valeur=v;

tmp->suivant=L;
L=tmp;
```

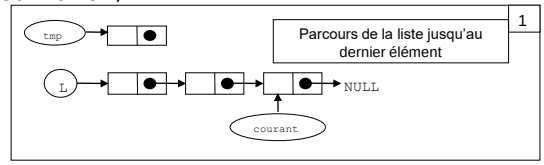


32

Les listes chainées

- Ajouter un élément à la fin d'une liste

```
struct  Liste   *tmp,*courant;
courant = L;
tmp =(struct Liste*)malloc(sizeof(struct Liste));
printf("ajouter :");
scanf("%d",&v);
tmp->valeur=v;
while(courant->suivant != NULL){
    courant = courant->suivant ;
}
```

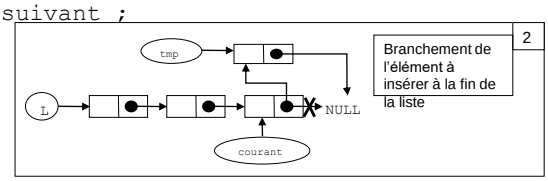


33

Les listes chainées

- Ajouter un élément à la fin d'une liste

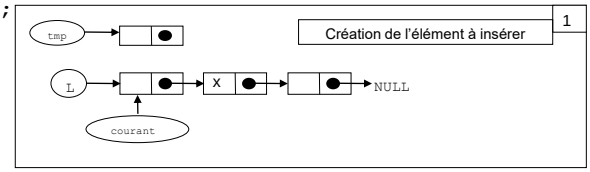
```
struct  Liste    *tmp,*courant;
courant = L;
tmp =(struct Liste*)malloc(sizeof(struct Liste));
printf("ajouter :");
scanf("%d",&v);
tmp->valeur=v;
while(courant->suivant != NULL){
    courant = courant->suivant ;
}
tmp->suivant=NULL;
courant->suivant=tmp ;
```



Les listes chainées

- Ajouter un élément au milieu d'une liste

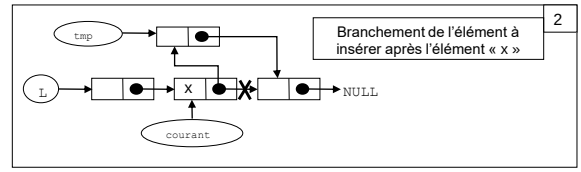
```
struct  Liste     *tmp , *courant;
courant = L;
tmp =(struct Liste*)malloc(sizeof(struct Liste));
printf("ajouter :");
scanf("%d",&v);
tmp->valeur=v;
```



Les listes chainées

- Ajouter un élément au milieu d'une liste

```
struct  Liste    *tmp , *courant;
courant = L;
tmp =(struct Liste*)malloc(sizeof(struct Liste));
printf("ajouter :");
scanf("%d",&v);
tmp->valeur=v;
```




```
while(courant != NULL && courant->valeur != x)
{   courant = courant->suivant ;  }
tmp->suivant = courant->suivant;
courant->suivant=tmp ;
```


Les listes chainées

- Supprimer un élément d'une liste : **Principe général**

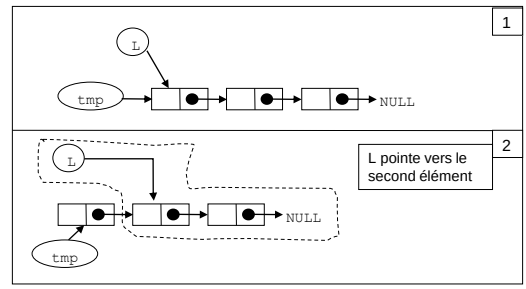
1. Pointer vers l'élément *x* à supprimer par un nouveau pointeur *px*
2. Il faut que l'élément qui est avant *x* pointe vers l'élément qui est après *x*
3. Libérer l'espace mémoire occupé par l'élément x en utilisant la fonction `free( )`
   Dans ce cas : `free(px)`
4. Décrémenter la variable qui stocke la taille de liste

- *Attention! Il ne faut pas casser un lien avant d'établir un autre*

Les listes chainées

- Supprimer un élément au début d'une liste

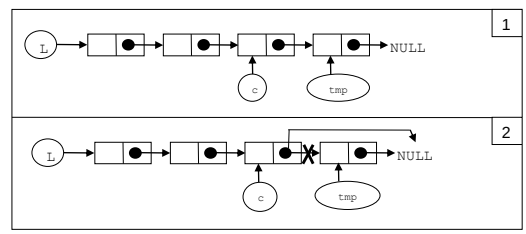
```
//... Après déclarations
tmp = L ;
L = L->suivant;
free(tmp);
```



Les listes chainées

- Supprimer un élément à la fin d'une liste

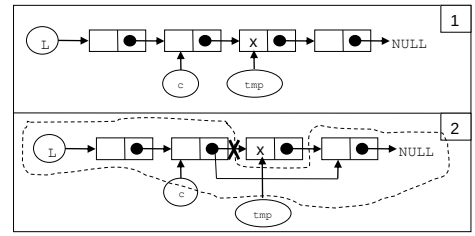
```
struct  Liste   *tmp,*c;
c = L;                      // c pointe vers le 1er élément
tmp = L->suivant ;          // tmp pointe vers le 2ème
while( tmp->suivant ){      // <=> tmp->suivant != NULL
    tmp = tmp->suivant ;
    c = c->suivant ;
}
c->suivant = NULL;
free(tmp) ;
```



Les listes chainées

- Supprimer un élément au milieu d'une liste

```
c = L;
while(c != NULL &&
    c->suivant != NULL &&
    c->suivant->valeur != x)
    {  c = c->suivant ;  }

if(c->suivant->valeur == x){
    tmp = c->suivant;
    c->suivant = tmp->suivant;
    free(tmp);
}
```



Allocation dynamique

Tableaux	Listes chainées
• Pour déclarer, il faut prévoir une taille (connue dès la déclaration)	• Pour déclarer, il suffit de créer un pointeur qui va pointer sur le premier élément (aucune taille n'est donc à spécifier)
• Il est possible d'atteindre directement la case **i**	• Impossible d'accéder directement à l'élément **i**
• Stockage contigu	• Stockage non contigu
• Création de nouveau tableau lors de l'ajout ou de la suppression	• ajouter, supprimer, ou intervertir des éléments sans avoir à recréer la liste

Conclusion sur les listes chaînées

- Structure orientée vers les traitements séquentiels
- Peut être manipulée de façon itérative ou de façon récursive
- L'implantation chaînée permet des ajouts et des suppressions sans déplacement
- Mais l'accès par position est proportionnel à la longueur de la liste
- Le coût d'un algorithme est évalué en place occupée et en nombre de pointeurs parcourus ou affectés
- A utiliser chaque fois que les mises à jour sont plus importantes que les consultations

Les listes doublement chainées

- Selon le même principe…

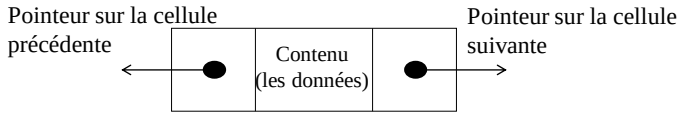
```
struct cellule {
  … // champs correspondant au contenu

  struct cellule *suiv ;
   //pointeur sur l'élément suivant
  struct cellule *prec ;
   //pointeur sur l'élément précédent
} ;
```

- On parle de **liste doublement chainée**

Les listes doublement chainées

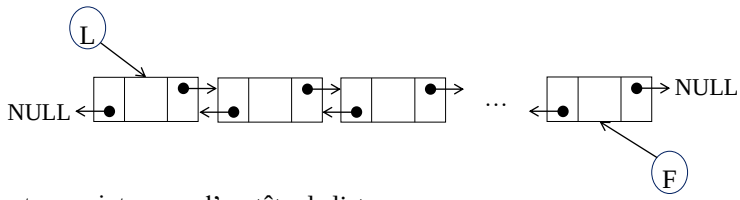
- A la différence des listes simplement chaînées, les maillons d'une liste doublement chaînée possèdent encore un pointeur sur l'élément qui les précèdent

```
struct cellule {
  ... // contenu
  struct cellule *suiv ;
  struct cellule *prec ;
} ;
```



Les listes doublement chainées

- Exemple de liste doublement chaînée



- L est un pointeur sur l'en-tête de liste
- F est pointeur sur la queue de liste

Les listes doublement chainées

- **Manipulations usuelles :**
  - Créer une liste, éventuellement vide
  - Parcourir une liste
  - Calculer le nombre d'éléments (la taille d'une liste)
  - Ajouter un élément à une liste (début, fin, milieu)
  - Supprimer un élément d'une liste (début, fin, milieu)
  - Trier les éléments d'une liste
  - Détruire une liste
  - …

Voir les codes sources en C

Les piles et les files

LIFO

Last In First Out

FIFO

First In First Out

Les piles et les files

- Les piles (stack) et les files (queue) constituent deux structures de données particuliers.
- Elles permettent de stocker diverses données comme pourrait le faire un tableau mais en respectant un critère particulier pour les gérer.
- Les piles sont utilisées pour stocker les valeurs des variables locales et peuvent être utilisées dans des algorithmes d'évaluation d'expressions mathématiques.
- Les files sont utilisées généralement pour mémoriser des données en attente de traitement.

48

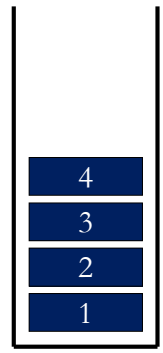
Les piles

- *Liste* dans laquelle les ajouts et suppressions n'ont lieu que sur une même ***extrémité*** appelée ***sommet de pile***.
- *Exemple* : pile d'assiettes, piles de livres, …
- ***Structure LIFO*** : le dernier élément entré (Last In) est le premier sorti (First Out).
- ***Utilisation :*** sauvegarder temporairement des informations en respectant leur ordre d'arrivée, et les réutiliser en ordre inverse
- ***Principe de fonctionnement :***
  - Dernier arrivé, premier servi
  - Ajout et retrait au sommet de la pile

49

Les piles

- La pile est un espace où on peut accumuler des éléments.
- On ne peut retirer que le dernier élément ajouté.



Les piles

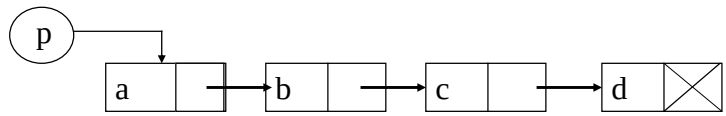
- Fonctions usuelles :
  - Créer une pile
  - Tester si une pile est vide
  - Compter le nombre d'éléments d'une pile
  - Retourner la valeur du premier élément (valeur du sommet)
  - Empiler : ajouter un élément
  - Dépiler : supprimer un élément

Les piles

- créerPile : initialise une pile à vide ⇒ doit être appelée avant toute utilisation d'une pile.
- pileVide, valeurSommet , empiler et dépiler ne sont pas définies sur une pile dont la valeur est indéterminée.
- valeurSommet et dépiler ne sont pas définies sur une pile vide.

52

Les piles



- Ecrire une fonction qui compte le nombre d'éléments d'une pile donnée.

```
int Nb_elements(Pile *p){
    ...
}
```


53

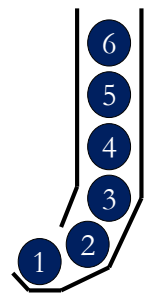
Les files

- Correspond à la notion usuelle de **file d'attente**: file d'attente à un guichet, file d'impression … « **queue** »…
- Liste dans laquelle les ajouts se font à une extrémité (fin de file) et les suppressions à l'autre (tête de file).
- ***Structure FIFO*** : le premier élément entré (**F**irst **I**n) est le premier sorti (**F**irst **O**ut).
- ***Utilisation*** : sauvegarder temporairement des informations en respectant leur ordre d'arrivée, et les réutiliser en ordre d'arrivée
- ***Principe de fonctionnement :***
  - Premier arrivé, premier servi
  - Ajout en queue et retrait en tête de la file

54

Les files

- La file est un espace où on peut accumuler des éléments.
- On ne peut retirer que le plus vieux des éléments ajoutés



55

Les files

- Fonctions usuelles :
  - Créer une file
  - Tester si une file est vide
  - Compter le nombre d'éléments d'une file
  - Retourner la valeur du premier élément
  - Enfiler : ajouter un élément (à la fin)
  - Défiler : supprimer un élément (au début)

56

Les files

- créerFile : initialise une file à vide ⇒ doit être appelée avant toute utilisation d'une file.
- fileVide, valeurPremier, enfiler et défiler ne sont pas définies sur une file qui n'a pas été créée au préalable.
- valeurPremier et défiler ne sont pas définies sur une file vide.

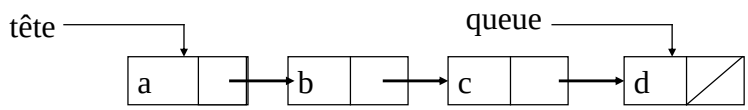
57

Les files

- Les systèmes d'exploitation ont recours à la structure de données « file d'attente » pour gérer l'accès à une ressource partageable :
  - file des requêtes d'impression (serveur d'impression)
  - file des processus en attente d'un processeur (ordonnanceur)

58

Les files



- Ecrire une fonction qui ajoute un élément à une file.
- Ecrire une fonction qui supprime un élément d'une file.

59