République Algérienne Démocratique et Populaire

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Département Informatique

Les Listes Chaînées-1

Cours Algorithmique de 1ere Année MI

Présenté par : Dr. B. BESSAA

Un petit retour en arrière !

On a vu que lorsqu'un ensemble de variables ont les mêmes propriétés et concernent un même objet (par exemple : les notes d'un étudiant), on les regroupe dans une seule variable de type **Tableau**. Ce qui facilite la manipulation des éléments de cet ensemble.

On a vu aussi, dans le cours des pointeurs, qu'à une variable de type **Tableau**, on fait correspondre **une seule adresse** : c'est l'adresse de **base**. Et à partir de cette adresse, on peut retrouver l'adresse (**@**) de n'importe quel élément du tableau juste en ayant son indice (**I**), en utilisant la fameuse formule : **@ = N*(I-1)+@1**
où **@1** est l'adresse de base

Vous êtes d'accord ? Vous vous souvenez de ça ?

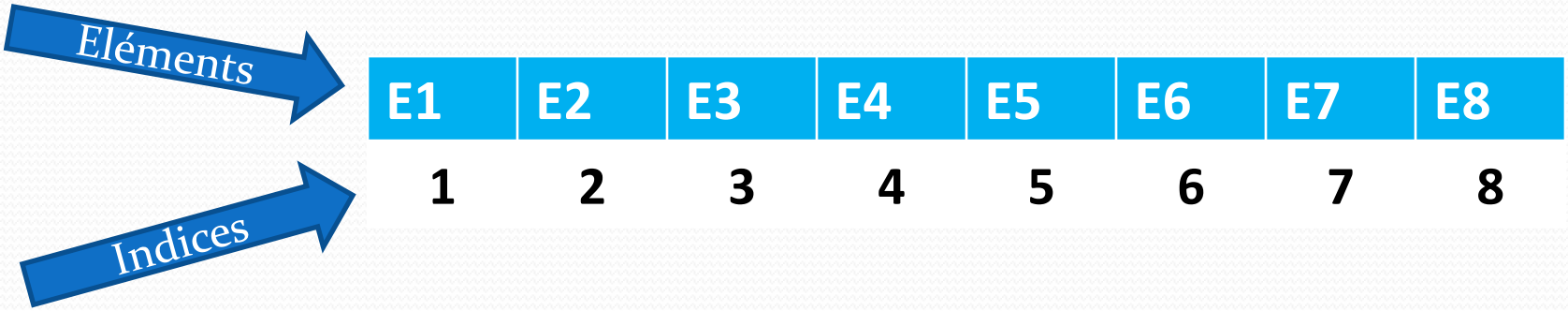
Oui, oui on vient juste de le voir.

Bien, alors comment cela est possible ?

Et bien ceci est possible, parce que les éléments du tableau sont stockés en mémoire l'un **à coté de** l'autre, ils sont **voisins**, il sont **adjacents**. On dit que les éléments du tableau sont **contigus**.

Au fait c'est une manière de **représentation** des données, on parle de la **représentation contigüe**.

L'avantage principal de la représentation **contigüe**, est qu'à partir d'une adresse de base, on peut **accéder directement** à un élément de l'ensemble juste en ayant sa **position** (son **indice**). Ce qui permet **d'accélérer** l'accès aux données.



Mais malheureusement, la représentation contigüe possède deux inconvénients majeurs.

1- La taille de la structure de donnée

En effet, dans une représentation contigüe, on doit spécifier la taille de la structure de donnée (la taille **max** du tableau). Et donc une fois le tableau plein, on ne peut pas ajouter des éléments.

Mais on peut utiliser un tableau dynamique. Et dans ce cas on n'a pas besoin de spécifier la taille max du tableau.

Bonne réflexion, oui, avec un tableau dynamique, on n'a pas besoin de spécifier la taille dans la **partie déclaration**, mais au moment de l'allocation dynamique (en exécution), il faut **préciser** la taille. Donc même avec un tableau dynamique la taille est limitée.

Et quel est le deuxième inconvénient ?

Le deuxième est **aussi important** que le premier.

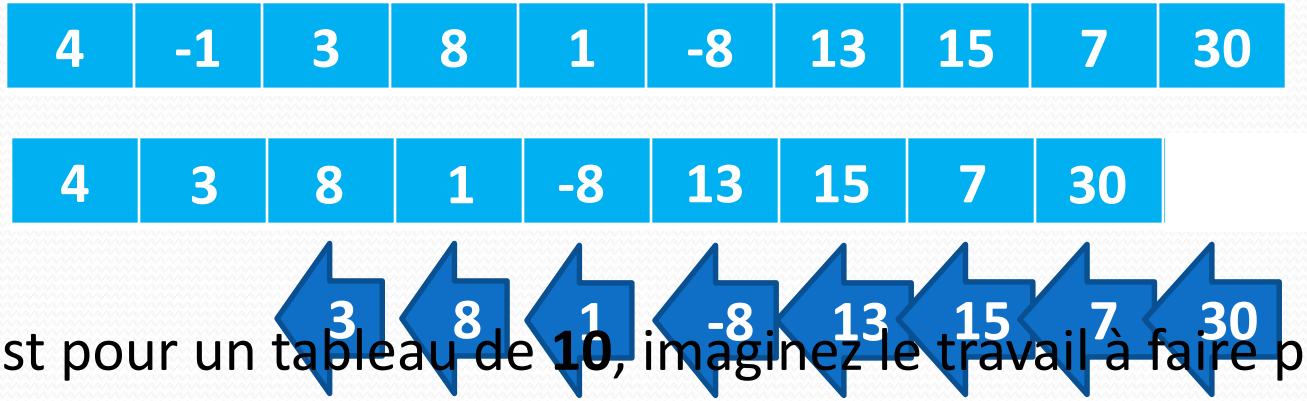
2- La mise à jour des données

2- La mise à jour des données

Alors si jamais on veut **supprimer** un élément ou **insérer** dans le mesure du possible (si l'espace le permet), on doit faire toute une gymnastique qui prend beaucoup de temps.

Oui on doit faire un décalage de tous les éléments.

Trèèèès bien, voici un exemple:
Soit un tableau de **10** éléments. Pour supprimer le deuxième élément, on doit décaler tous les éléments à partir du troisième d'une case à gauche (donc **8** décalages), ensuite on réduit la taille exacte (N) du tableau



Alors ça c'est pour un tableau de **10**, imaginez le travail à faire pour un tableau de **1000** **???!!!!** La même chose se fait pour l'insertion.

Donc que ça soit Statique ou Dynamique

La représentation **Contigüe** possède un avantage très intéressant (l'accès **rapide** aux données), mais elle a un inconvénient **majeur** dans le cas d'une mise à jour des données.

Cette manière de représentation est **conseillée** dans les problèmes où la mise à jour est **rare** : conception d'un dictionnaire par exemple.
Par contre, elle est **déconseillée** dans les cas où la mise à jour est **fréquente**.

Et alors, est-ce-qu'il y a solution à ce problème ?

Bien, quand on veut trouver des solutions à des problèmes pareils, il faut viser les inconvénients et essayer de trouver des remèdes. Là on a dit qu'il y a deux inconvénients. On va essayer de les éliminer un par un. Commençant par le plus important: le deuxième.
Essayons donc de trouver une solution pour éviter les décalages pour les opérations de suppressions ou d'insertions.

Le problème du décalage est dû au fait qu'il y a un **ordre** entre les éléments qu'il faut **respecter**, c'est un **lien implicite** entre les éléments. Alors si on veut éviter ce décalage, il faut **casser** cet ordre. Mais si on le casse on **perd** le lien entre les éléments. La solution est de rendre ce lien **explicite**.

Non, on n'a rien compris !!!

J'explique, soit cinq personnes alignés l'un après l'autre, et chacun possède un numéro.

Quel le numéro du **deuxième** ?

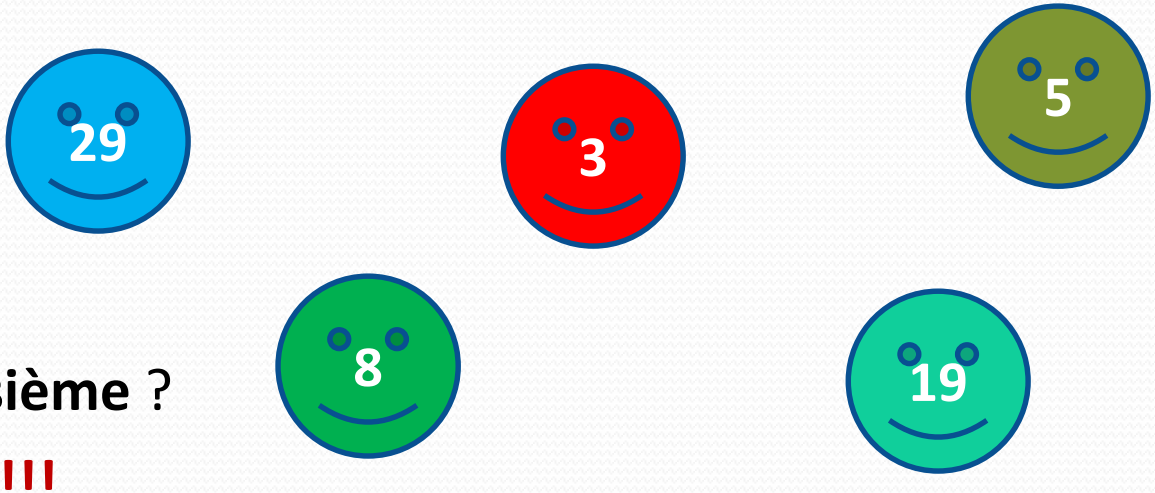
C'est 19.

Bien, quel le numéro du **quatrième** ?

C'est simple 29.

Donc vous avez pu répondre facilement car vous **voyez** l'ordre entre ces personnes, il suffit de compter.

Alors maintenant je vais **changer** la situation, et je présente les mêmes personnes de cette **manière**



Quel le numéro du **troisième** ?

Euuuuh, on ne sait pas !!!

Oui, et vous ne pouvez pas le savoir, parce qu'on a **cassé l'ordre initial** entre les personnes qui était visuel (**implicite**).

Et pour que vous puissiez répondre à cette question, il vous faut cette information. Mais comme elle n'est pas apparente, il faut la rendre claire (**explicite**)

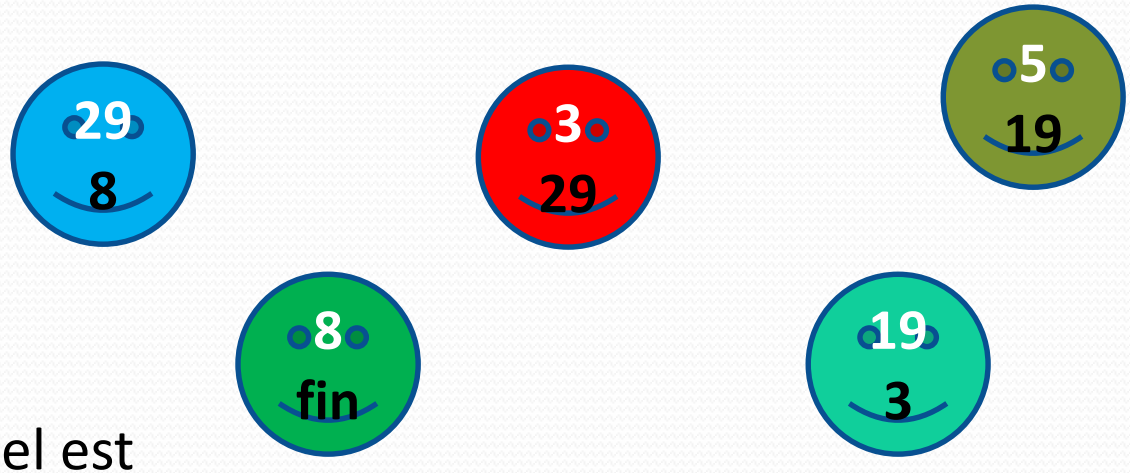
Et comment ?

On va voir.

Alors maintenant je vais **demander** à chacun de ces personnes de **se souvenir** du numéro de son **voisin** gauche. Et moi je ne garde que le numéro du **premier**, c'est le numéro **5**.

Entre nous, je vous rappelle l'ordre initial, c'était : **5-19-3-29-8**

Dans ce cas, la nouvelle situation sera:

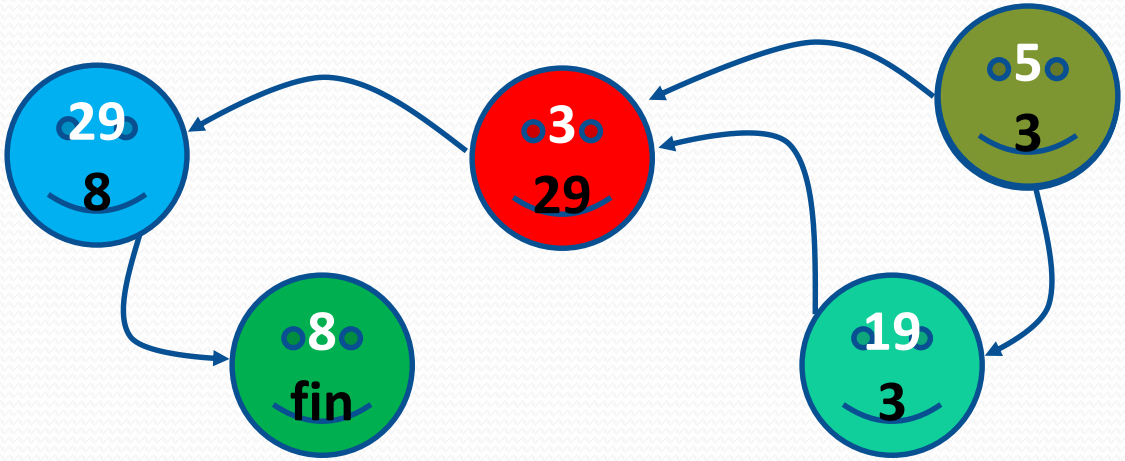


Je repose la question, quel est le numéro du **troisième** ?

Alors, le premier c'est 5, son voisin le deuxième, c'est 19 ensuite son voisin le troisième c'est le numéro 3.

Bien, vous voyez donc, maintenant le lien devient **explicite, vous le voyez**.

En effet, dans la nouvelle situation, il existe un **lien de chainage explicite** entre ces personnes:



Oui, mais comment ça résout le problème du décalage ?

C'est simple, si on veut supprimer le deuxième par exemple, il suffit de dire au premier que **ton voisin n'est plus 19** mais le voisin de **19** qui est **3**. Et donc on aura un nouveau chainage sans faire aucun décalage.

On appelle cette représentation : La représentation Chainée

La **Représentation Chainée** d'un ensemble d'éléments ordonnés, consiste à **ajouter** à chaque élément une **information** sur son **suivant**.

Donc chaque élément sera représenté par deux parties (**Enregistrement**)

1- Information : qui décrit l'élément lui-même.

2- Suivant : qui donne information (code) sur le suivant.

Et justement, les **structures de données** utilisant ce mode de représentation, s'appellent:

Les Listes Chaînées

Il existe plusieurs types de listes chainées, **linéaires** et **non linéaires** (**LLSC, LLDC, LLCC, Arbres, Graphes**,…), nous allons nous intéresser dans ce cours aux (**LLSC**):

Listes Linéaires Simplement Chaînées

Une **LLSC** est une suite ordonnée d'éléments de (maillons) de même type, reliés entre eux par un chainage **simple** (dans un seul sens).

Un **maillon** (élément) est un enregistrement composé de deux champs:

1- **Information** 2- **Suivant**

On peut toujours exploiter les tableaux (**organisation contigüe**) pour représenter une **LLSC** (tableau **d'enregistrements**), dans ce cas le suivant sera l'indice de l'élément suivant dans le tableau. Le dernier élément aura l'indice **0** pour son suivant. La liste sera définie par l'indice du **premier** élément.

Exemple:

Soit la liste suivante composée de **6** éléments, avec le premier élément qui se trouve à l'indice **4**:

	1	2	3	4	5	6	7	8	9
Information	5.25	8.13	10.1	18.2	5.1		-1.8		
Suivant	7	0	5	1	2		3		

La représentation contigüe équivalente est:

18.2	5.25	-1.8	10.1	5.1	8.13			

Mais dans ce cas on aura toujours le problème de la taille limitée du tableau !!!

Effectivement, en plus son implémentation est un peu compliquée. Donc cette représentation n'est pas intéressante, en passe alors à l'élimination du **deuxième** inconvénient de la représentation contigüe qui était justement la taille limitée du tableau.

Alors là c'est plus simple. Pensons au chainage qu'on a fait ?!

Les pointeurs ???

Exactement, on voit déjà dans le champ **suivant** du maillon de la liste, la notion de l'adresse de l'élément suivant, donc un pointeur vers l'élément suivant.

En utilisant les pointeurs, une liste sera définie par un **pointeur** sur le **premier** maillon (élément), et le **suivant** sera lui aussi un **pointeur** vers un maillon (élément suivant). Et donc, théoriquement, la taille ne sera pas limitée.

Déclaration d'une liste (LLSC)

On a deux notations possibles pour déclarer une liste:

Notation 1:

```
Type
<NomMaillon> = Enregistrement
                    <IdInfo>:<TypeElt>;
                    <IdSuivant>: ^ <NomMaillon>;
                Fin;
```

<TypeElt>: est le type de l'information qui peut être simple ou structuré.

Avec cette notation, les variables représentant des listes seront déclarées par des pointeurs vers <NomMaillon>.

Exemple

```
Type    Eliste =  Enregistrement
                    Info : reel;
                    Suiv : ^ Eliste;
                  Fin;
Var     L,Q : ^Eliste;
```

Notation 2:

```
Type    <NomPtr> = ^ <NomMaillon>;
        <NomMaillon> =  Enregistrement
                          <IdInfo> : <TypeElt>;
                          <IdSuivant> : <NomPtr>;
                        Fin;
```

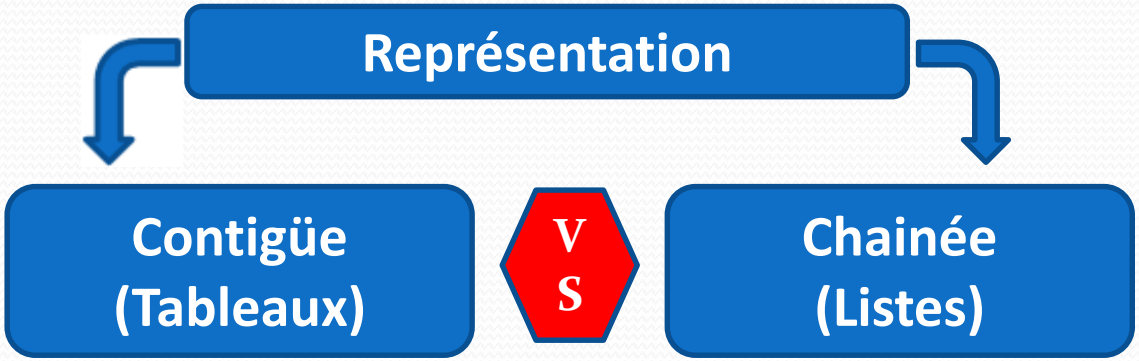
Avec cette notation, les variables représentant des listes seront déclarées comme étant des <NomPtr>.

Exemple

```
Type    Pliste = ^ Eliste;
        Eliste =  Enregistrement
                    Info : reel;
                    Suiv : Pliste;
                  Fin;
Var     L,Q : Pliste;
```

On peut utiliser l'une des deux, mais personnellement je préfère la deuxième, elle évite d'utiliser à chaque fois le symbole pointeur (**^**).

Enfin nous pouvons présenter un résumé sur la représentation des données avec une comparaison des performances :



Mémoire	Statique, Dynamique	Dynamique, (Statique)
Taille	Limitée	Non Limitée
Accès	Direct, Séquentiel	Séquentiel
MAJ	Complexe	Simple

A Suivre…

brbessaa@gmail.com