UNIVERSITÉ PRIVÉE DE MARRAKECH
UNIVERSITÉ RECONNUE PAR L'ÉTAT
جامعة معترف بها من طرف الدولة

Module: Structures de donnée

Chapitre 4: Les listes simplement chainées

F. OUAKASSE

Plan

- Définition
- Principe
- Liste chaînée Vs. Tableau
- Construction
- Insertion en tête de liste
- Parcours

Définition

Une liste chaînée est un **ensemble de cellules liées (maillons)** entre elles par des pointeurs. Chaque cellule est une structure contenant les champs suivants :

1- une **donnée**

2- un **pointeur suivant** sur la cellule suivante.

Il existe plusieurs types de listes chaînées dépendant de la manière dont on se déplace dans la liste :

1. les listes chaînées simples,
2. les listes doublement chaînées,
3. les listes circulaires.

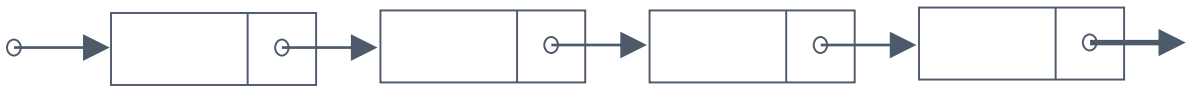
Principe

- Les listes chaînées représentent une façon d'organiser les données en mémoire de manière beaucoup plus flexible. Comme à la base le langage C ne propose pas ce système de stockage, nous allons devoir le créer nous-mêmes de toutes pièces
- le problème des tableaux est qu'ils sont figés. Il n'est pas possible de les agrandir, à moins d'en créer de nouveaux, plus grands. De même, il n'est pas possible d'y insérer une case au milieu, à moins de décaler tous les autres éléments.

Principe

Une liste chaînée est un moyen d'organiser une série de données en mémoire. Cela consiste à assembler des structures en les liant entre elles à l'aide de pointeurs. On pourrait les représenter comme ceci :

une chaîne de pointeurs



Principe

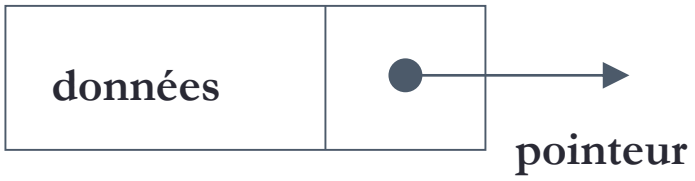
Notion de maillon

L'élément de base d'une liste chaînée s'appelle le maillon.

Il est constitué :

*** d'un champ de données (information) ;**

*** d'un pointeur vers un maillon (adresse maillon suivant).**



Principe

Maillon suivant

Le champ *pointeur vers un maillon* pointe vers le maillon suivant de la liste. S'il n'y a pas de maillon suivant, le pointeur vaut NULL.

Connaissant le pointeur sur le premier élément on peut connaitre tous les éléments de la chaines.

Principe

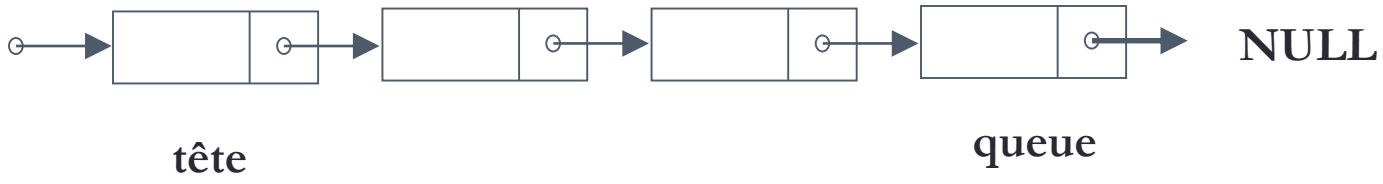
Notion de liste

Une liste simplement chaînée est un pointeur sur un maillon. Une liste vide est une liste qui ne contient pas de maillon. Elle a donc la valeur NULL.

Principe

La terminologie suivante est généralement employée :

* le premier maillon de la liste est appelé tête (début) ;
* le dernier maillon de la liste est appelé queue (fin).

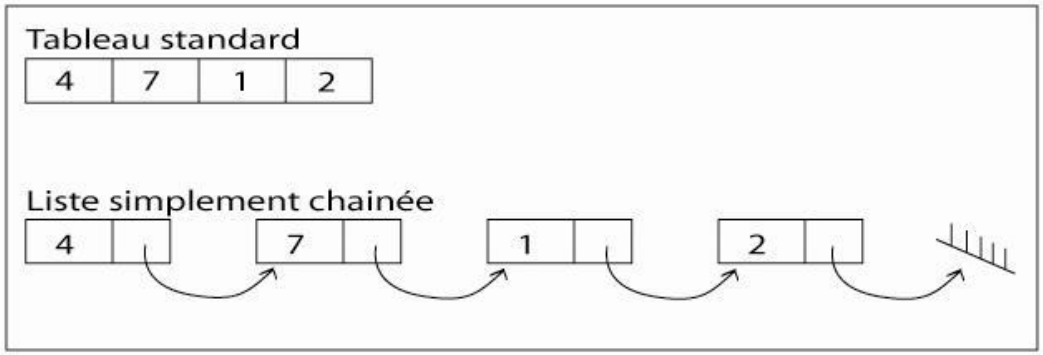


Rq : il se peut que les maillons qui composent la liste ne soit pas placées dans l'ordre en mémoire.

Liste chaînée Vs. Tableau

Dans une liste chaînée :

- la taille est inconnue au départ, la liste peut avoir autant d'éléments que votre mémoire le permet.
- Pour déclarer une liste chaînée il suffit de créer le pointeur qui va pointer sur le premier élément de votre liste chaînée, aucune taille n'est à spécifier donc.
- Il est possible d'ajouter, de supprimer, d'intervertir des éléments d'un liste chaînées en manipulant simplement leurs pointeurs.



Liste chaînée Vs. Tableau

Un pointeur vers un élément du même type appelé suivant. C'est ce qui permet de lier les éléments les uns aux autres : chaque élément « sait » où se trouve l'élément suivant en mémoire.

les cases d’une liste ne sont pas côte à côte en mémoire. C'est la grosse différence par rapport aux tableaux. Cela offre davantage de souplesse car on peut plus facilement ajouter de nouvelles cases par la suite au besoin.

Liste chaînée Vs. Tableau

Remarque :

Le maillon d'une liste ne sait pas quel est l'élément précédent, il est donc impossible de revenir en arrière à partir d'un élément avec ce type de liste. On parle de liste « simplement chaînée », alors que les listes « doublement chaînées » ont des pointeurs dans les deux sens et n'ont pas ce défaut. Elles sont néanmoins plus complexes.

Construction

Le maillon

Nous allons créer une liste chaînée de nombres entiers. Chaque élément de la liste aura la forme de la structure suivante.

Pour créer une liste chaînée, il faut d'abord déclarer une structure qui représentera un maillon (une cellule).

Construction

Exemple : déclaration d'une liste dont les données sont des int.

```
Typedef struct Cell
      {
       int donnee;     //définition des données
       struct Cell *suivant;  //pointeur sur la structure suivante
(de même type que celle qu'on est entrain de définir)
}TypeCellule;
```

On déclare ensuite la liste List avec un pointeur de type TypeCellule.

```
      TypeCellule *List;
```

Construction

- **Création d'un élément de la liste:**

```
typedef  struct Element Element;
struct Element
        {
         int nombre;
        Element *suivant;
        };
```

Rappel

- Les fonctions de gestion de la liste

Nous avons créé deux structures qui permettent de gérer une liste chaînée :

- Element, qui correspond à un élément de la liste et que l'on peut dupliquer autant de fois que nécessaire ;
- Liste, qui contrôle l'ensemble de la liste. Nous n'en aurons besoin qu'en un seul exemplaire.

Maintenant on aura besoin de :

- initialiser la liste ;
- ajouter un élément ;
- supprimer un élément ;
- afficher le contenu de la liste ;
- supprimer la liste entière.

Initialisation

- La fonction initialisation

```
main()
{
Element *teteliste = (Element*)malloc(sizeof(Element));
    // la tête de la liste
   Element *element = (Element*)malloc(sizeof(Element));
   if (teteliste == NULL || element == NULL)
   {     exit(EXIT_FAILURE);
   } // tester est ce que l'allocation est réussie
   element->nombre = 0;
   element->suivant = NULL;
   teteliste = element; // garder toujours la tête de la liste
```

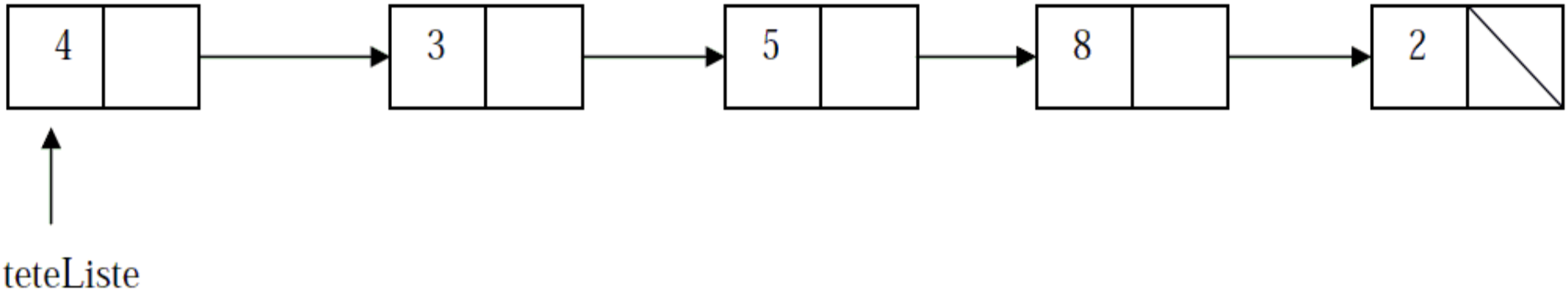
Ajout d'éléments

- Où va-t-on ajouter un nouvel élément ? Au début de la liste, à la fin, au milieu ?

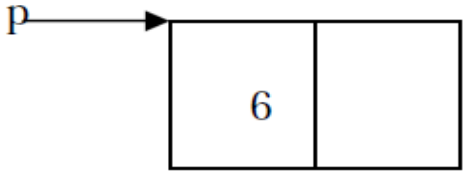
```
/* Création du nouvel élément */

Element *nouveau = (Element*)malloc(sizeof(Element));
    if (liste == NULL || nouveau == NULL)
    {      exit(EXIT_FAILURE);
    }
/* après avoir créer le nouveau élément on peut l'insérer au début/fin/milieu de la liste */
```

Insertion au début de la liste

Exemple : On suppose que l'on veut rajouter l'élément 6 au début de la liste suivante : teteListe

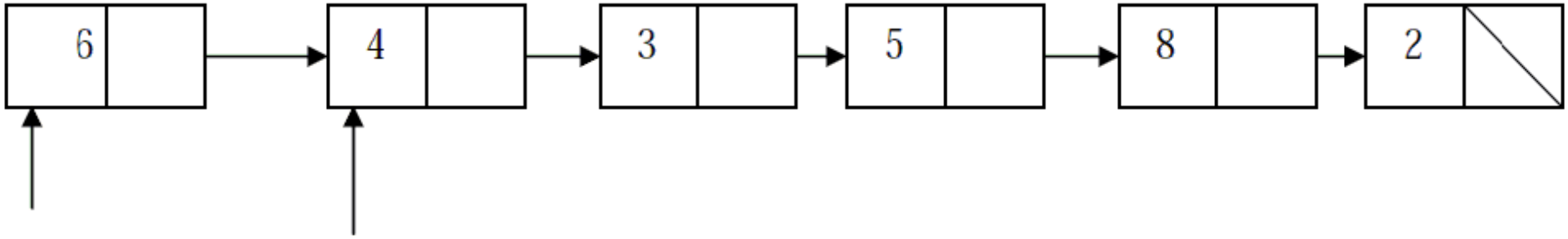


1) on commence par créer le nouvelle cellule et lui donner son contenu p

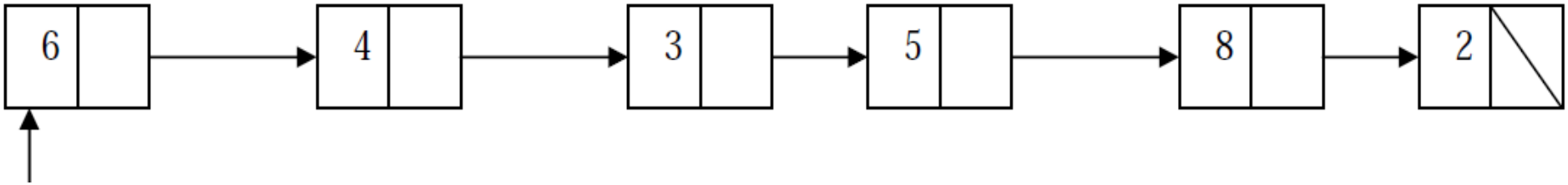


Insertion au début de la liste

2) le suivant de p est l'ancienne tête de liste



3) Le pointeur p devient tête de liste



Insertion au début de la liste

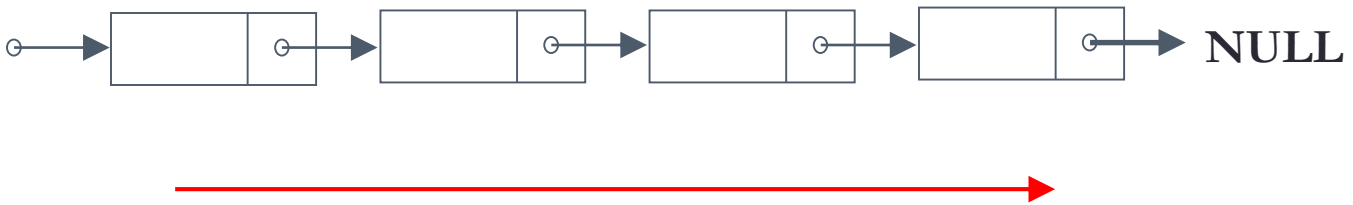
Pour insérer un nouvel élément au début de la liste, il faut créer le nouvel élément (déjà fait), puis faire pointer nouveau suivant sur la tête de la liste, enfin garder la tête de la liste qui prendra la valeur du nouvel élément:

```
nouveau->suivant=teteliste;
nouveau->nombre=5;
teteliste=nouveau;
```

Parcours

D'après son principe et sa construction, le parcours d'une liste simplement chaînée se fait uniquement du début vers la fin.

Illustrons ceci par l'affichage d'une liste simplement chaînée.



Parcours

L'idée de parcours d'une liste chaînée est de prendre un pointeur auxiliaire p. On fait pointer p sur la première cellule de la liste, puis le pointeur p passe à la cellule suivante (par une affectation p = p→suivant), etc. Le parcours s'arrête lorsque p vaut NULL.

Parcours

Exemple : affichage d'une liste chaînée.

```
        if (liste == NULL)
    { exit(EXIT_FAILURE);
    }
    Element *p = teteliste;// créer un pointeur p qui point sur le premier élément
    while (p!= NULL) // le parcours s'arrete lorsque p vaut null
    {
        printf("%d -> ", p->nombre);// tant que p is not null, le pointeur passe à la cellule suivante
        p = p->suivant;
    }
    printf("NULL\n");
```

Insertion en queue de la liste

L'ajout d'une cellule en queue de la liste est plus compliqué que l'insertion en tête de la liste. Elle nécessite un parcours de la liste pour rechercher l'adresse du dernier élément.

Insertion en queue de la liste

```
Element *nouveauFin; // créer le nouvel élément
 nouveauFin->nombre = 3;
 nouveauFin->suivant = NULL;// son suivant pointe sur null-> il sera le dernier élément de la liste
 p = teteliste; // créer un p pour parcourir la liste
 while (p->suivant != NULL) // parcourir la liste jusqu'au dernier élément
    { actuel = actuel->suivant;
    }
 p->suivant = nouveauFin; // stocker dans la dernière cellule le nouvel élément
```

Supression d'éléments

- Où va-t-on ajouter un nouvel élément ? Au début de la liste, à la fin, au milieu ?

```
if (teteliste == NULL)
  { exit(EXIT_FAILURE);
  }
if (teteliste!= NULL)
  {
     Element *aSupprimer = teteliste;
     teteliste = liste->suivant;
     free(aSupprimer);
  }
```

Libération de la mémoire

Pour libérer la mémoire d'une liste chaînée, il faut détruire chacune des cellules avec la fonction free.

```
Element * p; // créer un pointeur pour parcourir la liste
while (liste !=NULL)
        {  p=liste;
           p = p->suivant;  //Au fur et é mesure on parcourt la liste qu'on détruit les cellules
           free(p);      //destruction de la cellule
           }
liste = NULL;   //on réinitialise la liste à vide
```