Structures de données élémentaires

Listes chaînées

Définition

- Une liste $\mathcal{L}$ est une structure qui permet de stocker une suite d'éléments de même type. Cet ensemble possède les propriétés suivantes :
  - l'existence d'un premier élément que nous appelons Tête de la liste
  - l'existence d'un dernier élément que nous appelons Queue de la liste ;
  - L'existence pour tout élément, à l'exception de la queue, d'un Successeur ;
  - l'existence pour tout élément, à l'exception de la tête, d'un Prédécesseur.

Définition (suite)

$\mathcal{L} = (e_1, e_2, \ldots, e_{k-1}, e_k , e_{k+1}, \ldots , e_n)$;

- e_1 est Tête de la liste $\mathcal{L}$.
- e_n est Queue de la liste $\mathcal{L}$.
- e_{k+1} est le successeur de e_k dans $\mathcal{L}$.
- e_{k-1} est le prédécesseur de e_k dans $\mathcal{L}$.
- n est la longueur de la liste $\mathcal{L}$.
- k est le rang de e_k dans $\mathcal{L}$.

Rôle & Exemple

- On utilisera une liste pour stocker un nombre indéterminé d'éléments de même type dans un certain ordre.
- L'ordre pourra dépendre de la chronologie d'insertion des éléments, de la valeur des éléments insérés ou d'un quelconque autre critère fixé par l'utilisateur.
- Exemple : Un polynôme peut être représenté par une liste de monômes. Un monôme est défini par le coefficient et le degré. Ainsi, le polynôme $2X^{70}+1$ sera représenté par la liste : (2 , 70) , (1 , 0)

Description fonctionnelle

- Soit **Telement** le type de base des éléments de la liste et soit **Tliste** un ensemble d'éléments de type **Telement** doté d'une structure de liste .
- Les opérations qu'on peut effectuer sur les listes linéaires sont nombreuses, nous ne citons que les plus fondamentales

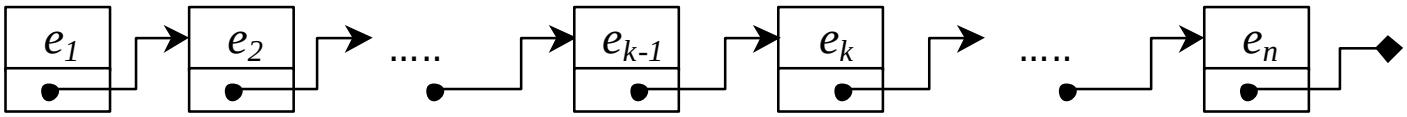
Description au niveau fonctionnelle

Nom opération	Domaine	Nature & Description
Construction		
CreerListe	-----→ **Tliste**	Création d'une liste vide
Modification		
InsereTete	**TListe** x **Telement**-→ **TListe**	Insérer un élément en début de liste
InsereQueue	**TListe** x **Telement**-→ **TListe**	Insérer un élément en fin de liste
Inserer	**TListe** x **Telement** x **Telement** →**TListe**	Insérer un élément après un élément donné
SupprTete	**TListe** -→ **TListe**	Suppression du premier élément
SupprQueue	**TListe** -→ **TListe**	Suppression du dernier élément
SupprElt	**TListe** x **Telement** -→ **TListe**	Détruire un élément donné
Utilisation		
ListeVide	**Tliste** -----→ booleen	Tester si la liste est vide
RangElt	**TListe** x **Telement** -→ Entier	Le rang d'un élément dans la liste
SuccElt	**TListe** x **Telement** -→ **Telement**	Successeur d'un élément donné
PredElt	**TListe** x **Telement** -→ **Telement**	Prédécesseur d'un élément donné
LongListe	**TListe** -----→ Entier	Longueur de la liste

Description au niveau logique

Une liste linéaire est soit vide, soit non vide.

- Une liste vide est une liste qui ne comporte aucun élément.
- Les éléments d'une liste non vide ne sont pas obligés d'être contiguës dans la mémoire.
  C'est pourquoi nous devons mettre en place un dispositif qui permet à chaque élément de désigner son successeur (ou son prédécesseur).

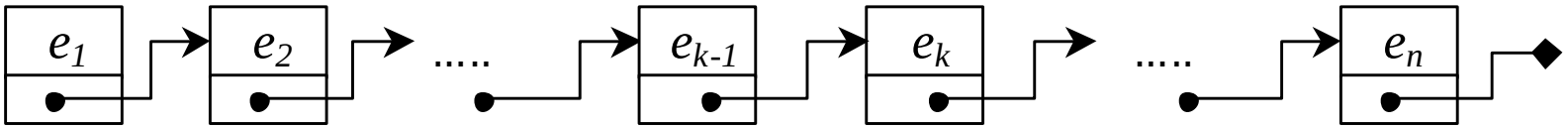


Description au niveau logique

- Nous utilisons le notion de **maillon**:

Un maillon étant une **structure** qui contient deux parties : la valeur de l'élément et un indicateur (pointeur ou indice) sur le maillon successeur.

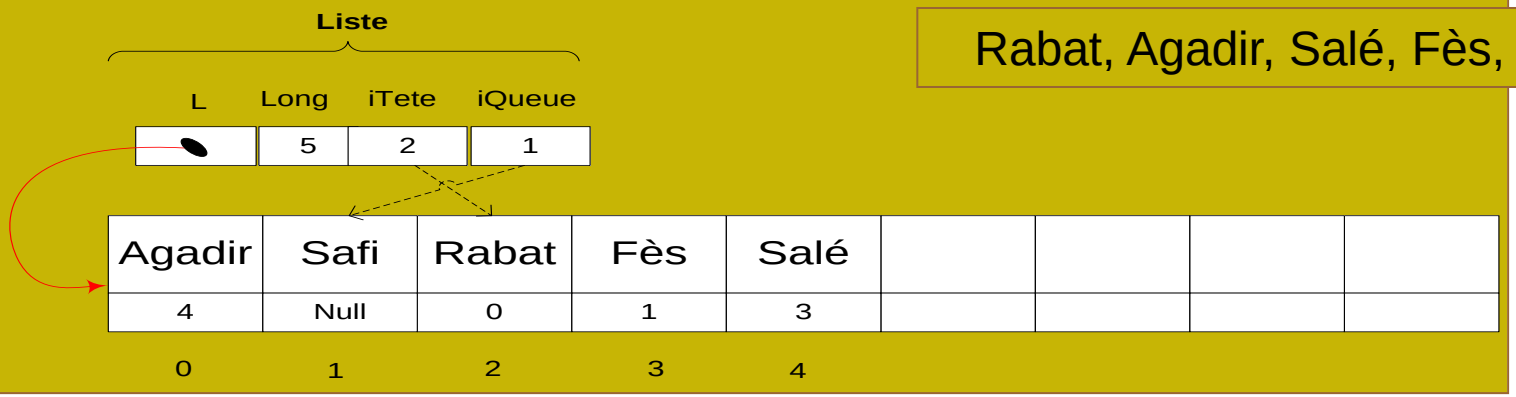
Une liste non vide est un ensemble de **maillons** (nœuds ou éléments).



Description au niveau physique :

a. Modélisation contiguë

- Cette méthode utilise un tableau de maillons.
- L'indice de Tête et celui de Queue doivent être stockés dans la structure .
- il est nécessaire d'utiliser Nmax la longueur maximale de la liste. Pour un tableau statique, Nmax sera une constante

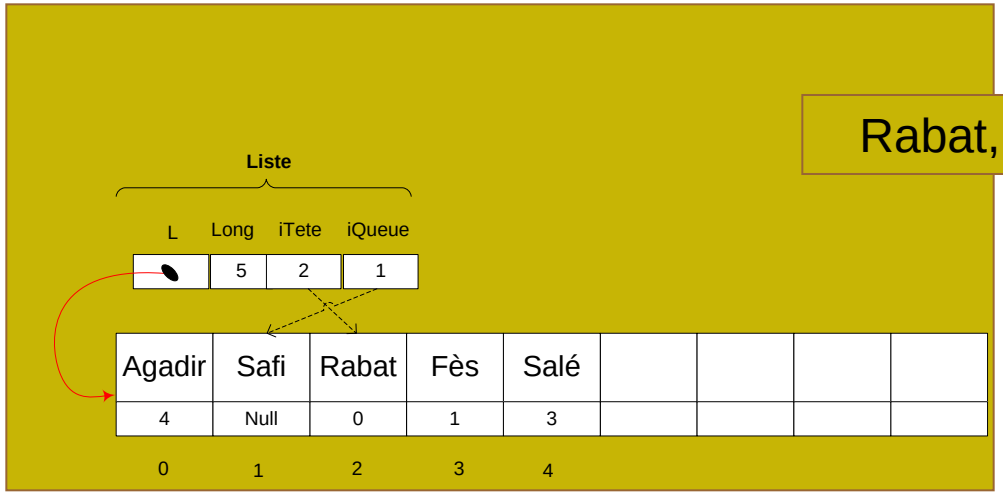


Description au niveau physique (en C) :

a. Modélisation contiguë statique

```
struct maillon {
    Telement e  ;
    int iSuivant  ;
    };
typedef struct maillon Tmaillon;
```

```
struct Liste {
  Tmaillon L[Nmax];
  int Long ;
  int iTete  ;
  int iQueue  ;
};
typedef struct Liste TListe ;
```



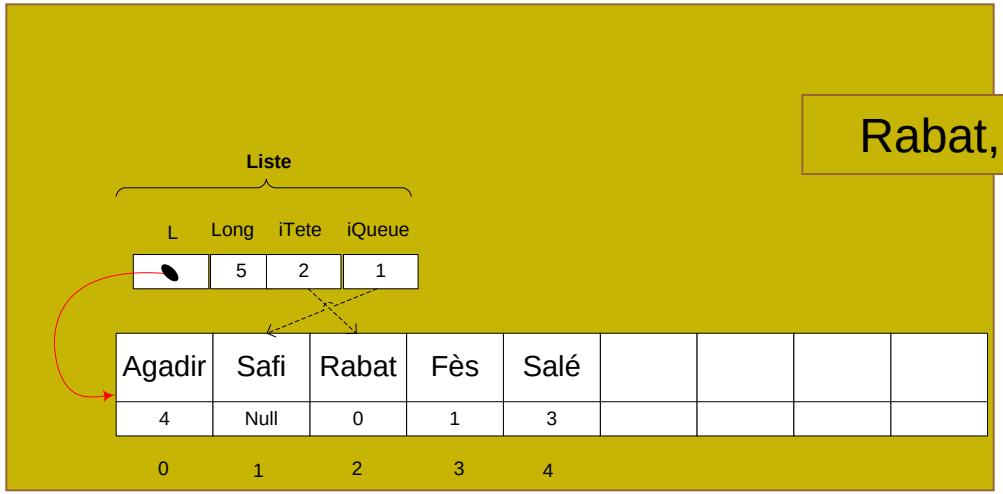
Rabat, Agadir, Salé, Fès, Safi

Description au niveau physique (en C) :

a. Modélisation contiguë dynamique

```
struct maillon {
    Telement e  ;
    int iSuivant  ;
    };
typedef struct maillon Tmaillon;
```

```
struct Liste {
  Tmaillon *L;
  int Long ;
  int iTete  ;
  int iQueue  ;
};
typedef struct Liste TListe ;
```



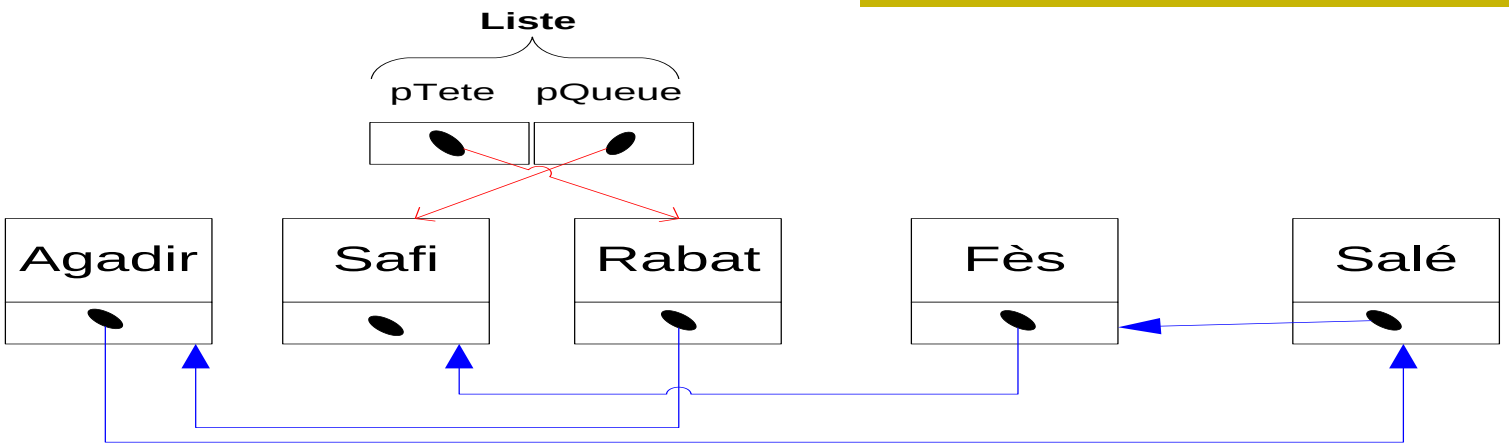
Rabat, Agadir, Salé, Fès, Safi

Description au niveau physique :

b. Modélisation chaînée

- Les maillons de la liste sont alloués dynamiquement chaque fois que cela est nécessaire.
- Dans ce cas, les maillons ne sont pas contiguës dans la mémoire.
- La liste est définie par l'adresse de la Tête et celle de la queue

Rabat, Agadir, Salé, Fès, Safi



Description au niveau physique (en C):

b. Modélisation chaînée

```
struct maillon {
        Telement e  ;
         struct maillon *pSuivant  ;
         };
typedef struct maillon Tmaillon;
```

```
struct liste {
            Tmaillon *pTete ;
            Tmaillon *pQueue ;
    };
typedef struct liste TListe;
```

Primitives sur les listes : Prototypes

- TListe *CreerListe() ;
- int ListeVide(TListe L) ;
- int LongListe (TListe L) ;
- Tmaillon *CreerMaillon(Telement valeur) ;
- int InserTete (TListe *L, Telement valeur) ;
- int InsererMaillon(TListe *L, Telement e, Tmaillon *position);
- void SupprTete(TListe *L);
- void SupprMaillon(TListe *L, Tmaillon *position);

Primitives sur les listes

Modélisation chaînée

```
TListe *CreerListe()
   {     Tliste *pL;
         pL=malloc(sizeof(Tliste));
         pL->pTete=Null;
         pL->pQueue= Null;
         return pL;
    }

int ListeVide(TListe L)
   {
        return  (L.pTete==Null) ;
    }
```

LongListe

```
int LongListe (TListe L)
 {
     int longueur=0
     Tmaillon *p =L.pTete ;
           while( p!=Null)
            {
                      longueur++;
                     p=p->pSuivant;
             }
  return longueur;
}
```

CreerMaillon

```
Tmaillon *CreerMaillon( Telement valeur )
    {
      Tmaillon *pm=malloc(sizeof(Tmaillon));
      if  (pm!=Null)
          {
          pm->e= valeur ;
         pm->psuivant= Null;
          }
    return pm;
}
```

InserTete

```
int InserTete (TListe *pL, Telement valeur) ;
    {
       Tmaillon *pm=CreerMaillon(valeur);
      Tmaillon *pT;
     if (pm==Null) return 0;
    pT=pL->pTete;
    pL->pTete=pm;
    pm->pSuivant=pT;
    return 1;
    }
```

Exercices:

- Donner la définition des fonctions :
  1. int InsererMaillon(TListe *L, Telement e, Tmaillon *position);
  2. void SupprTete(TListe *L);
  3. void SupprMaillon(TListe *L, Tmaillon *position);
- Proposer un programme qui permet
  1. de créer à partir du clavier une liste de nombres entiers,
  2. d'afficher les éléments de la liste à l'écran,
  3. La fréquence d'un nombre donné dans la liste.