Filière: MPI

**Module:** Programmation II

# Plan du cours

Chapitre 1. Les structures séquentielles (liste, pile, file)

Chapitre 2. Les structures arborescentes

# Plan du cours

Chapitre 1. Les structures séquentielles (liste, pile, file)

# 1-Introduction

• Une liste est une suite d'éléments repérés par leurs rangs (premier, deuxième, troisième, ….) .

Exemple:

L=(1, 5 , 10, 48)
L=("insat", "ensi", "supcom")

• L'ajout et la suppression d'un élément peut se faire à n'importe quel rang valide de la liste

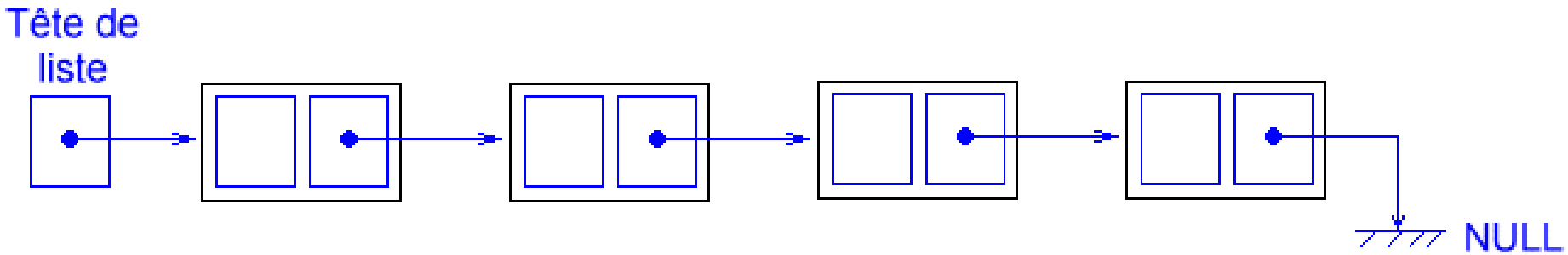
• Deux manières d'implémentation des listes:

1- Les tableaux
2- **Les listes chainées**

Les tableaux sont une première façon spontanée de représenter une liste. Les opérations d'insertion et de suppression d'éléments y sont toutefois malaisés.

Nous allons voir une autre façon de représenter une liste, à l'aide de données dynamiques et de pointeurs : les listes chainées.

**Une liste chainée est une structure dynamique formée de nœuds reliés par des liens.**



Les opérations de base qui peuvent être appliquées sont:

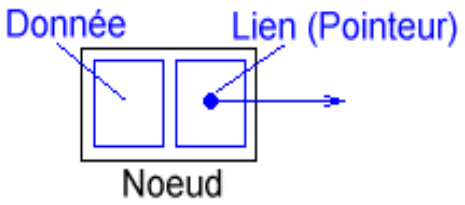
- Accéder à un nœud
- Insérer un nœud
- Supprimer un nœud

## 2- Représentation d'une liste chainée

L'élément de base d'une liste chaînée s'appelle le nœud. Il est constitué :

– d'un champ de données ;

– d'un pointeur vers un nœud.



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

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

**Définitions**
La terminologie suivante est généralement employée :
– le premier nœud de la liste est appelé **tête** ;
– le dernier nœud de la liste est appelé **queue**.

## 3-Définition en C

Définition en C de la structure Nœud

```
typedef struct Nœud
{
 int valeur;
struct Nœud * suivant;
}  Noeud;
```

Définition en C d'une liste chainée (liste est un type pointeur vers la structure)

```
Typedef  Noeud * liste;
liste    list;
```

Deux manières de définir une liste chainée

1- Liste chainée itérative
2- Liste chainée récursive

## 4-Liste chainée itérative

Principe: Définir la liste chainée itérative en prenant comme opérations de base:

- L'accès à toutes les positions des nœuds
- L'insertion
- La suppression

## Les opérations sur une liste chainée itérative

***Tester si une liste est vide*** : *int estvide(liste );*
***Insérer un élément à la tête d'une liste*** : *liste inserer_tete(liste list, int val);*
***Insérer un élément à la queue d'une liste*** : *liste inserer_fin(liste list, int val);*
***Supprimer un élément ayant la valeur val***: *liste supprimer(liste list, int val);*
***Déterminer la longueur d'une liste*** : *int longueur(liste list);*
***Afficher les éléments d'une liste*** : *void affichage(liste list);*

## - programmes en C des opérations d'une liste chainée itérative

```
int estvide(liste list)
{
if (list==NULL)
 return (1);
else
return (0);
}
```

```
int longueur(liste  list)
 {
 liste p_maillon = list;
 int lon = 0;

 while (p_maillon != NULL)
 {
 lon++;
 p_maillon = p_maillon → suivant;
 }
return (lon);
 }
```

```
void affichage(liste list)
 {
 liste p_maillon = list;

if(p_maillon ==NULL)
printf("liste vide\n") ;
else
{
 while (p_maillon !=NULL)
 {
 printf("%d ", p_maillon → valeur);
 p_maillon=p_maillon → suivant ;
 }
}
}
```

```
liste inserer_tete(liste list, int valeur)
{ liste list_i;
if ((list_i = (liste)malloc(sizeof(Noeud))) == NULL)
{ printf ("erreur allocation");
exit( l );
}
list_i →valeur = valeur;
list_i →suivant = list;
return(list_i);
}
```

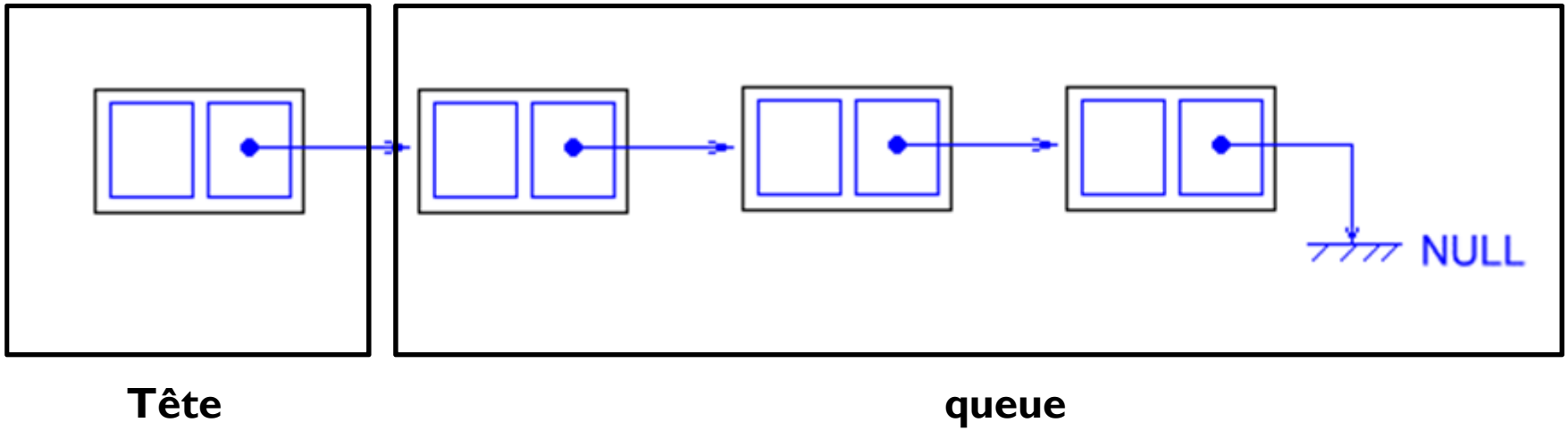
```
liste inserer_queue(liste list, int valeur)
{
liste list_i, list_move;
if((list_i = (liste )malloc(sizeof(Noeud))) == NULL)
{ printf(" erreur allocation "); exit( 1 ); }
list_i → valeur = valeur;
list_i → suivant= NULL;
if(list == NULL)
{
return(list_i);
}
else {
list_move= list;
while (list_move → suivant!=NULL)
list_move=list_move → suivant;
list_move → suivant = list_i;
return(list);
}
}
```

```
/* supprimer un element de la liste ayant le champ valeur égale à valeur */
 /* retourne la nouvelle liste creée */
liste supprimer(liste list, int valeur)
{ liste list_move,  before_list_move;
before_list_move = NULL;
list_move=list;
while ((list_move !=NULL) && (list_move → valeur !=valeur))
{before_list_move=list_move;
list_move=list_move → suivant;
}
if(list_move == NULL)
{ return(list); }
else { if(before_list_move == NULL)
{ list = list_move → suivant;
free(list_move);
return(list); }
else {
before_list_move →suivant = list_move → suivant;
free(list_move);
return(list); } } }
```

## 5-Liste chainée récursive

Principe: Définir la liste chainée récursive en prenant comme opération de base:

- Tete: rendre la valeur du premier nœud de la liste.
- queue: rendre une liste sans le premier nœud (suppression de la tête).
- Ajout d'un nœud à la tête d'une liste.



**Les opérations sur une liste chainée récursive**

***créer une liste*** : *liste creer(void);*
***Test si une liste est vide*** : *int estvide(liste );*
***Déterminer la valeur de la tête de la liste récursive :*** *int tete (liste list)*
***Déterminer la queue d'une liste récursive:*** *liste queue (liste list)*
***Déterminer la longueur d'une liste*** : *int longueur_recursif(liste list);*
***Afficher les éléments d'une liste*** : *void affichage_recursif(liste list);*

## - programmes en C des opérations d'une liste récursive

```
liste creer(void)
{
return NULL ;
}
```

---

```
int estvide(liste list)
{
if (list==NULL)
 return (1);
else
return (0);
}
```

```
int tete (liste list)
{
if(list!=NULL)
return(list→valeur);
else
{
Printf("liste vide\n »);
Exit (1);
}
}
```

```
liste queue (liste list)
{
liste aux=list;
if(aux!=NULL)
{
aux=aux →suivant;
return(aux);
}
Else
{
Printf("liste vide\n »);
Exit (1);
}
}
```

```
int longueur_recursif(liste  list)
{
if(list==NULL)
return 0;
else
return (1+ longueur_recursif(queue(list)));
}
```

---

```
   void affichage_recursif(liste  list)
{
if(list!=NULL)
{
printf("%d\n",  tete(list));
affichage_recursif(queue(list))
}
else
printf("\n");
}
```

```
int  fx(liste l, int val)
 {
   liste p=l;
   if (p == NULL)
   {
            return (0);
    }
    else if (p →valeur==val)
    {
            return (1);
    }
    else
    {
            fx(p →suivant, val);
    }
}
```

```
liste  fy(liste l, int val)  {
   liste R;
   if (l == NULL) {
       return (l);
    }
    if (l→valeur==val) {
       R= l;
       l= l→suivant;
       free(R);
       return (l);
    }

    else {
        l→suivant= fy (l→suivant,val); %chaînage de la liste
        return  (l);   %têtes de la liste
    }
}
```

## Vérification récursive de l'appartenance à une liste

```
int  contientRec(liste list, int v)
{
  liste p=list;
  // Si la liste est vide, elle ne contient pas v
  if (p == NULL)
    return 0;
  else if (p->valeur== v) // Sinon, si elle commence par v, alors elle le contient
    return 1;
  else // Sinon, elle contient v si la suite de la liste le contient
    return contientRec(p->suivant, v);
}
```

## Suppression

```
liste  rm(liste l, int n)  {
if (l == NULL) {
      printf("\nLa valeur n'est pas dans la liste");
      return l;
   }
   if (l->valeur==n) {
      liste R;
      R= l;
      l= l->suiv;
      free(R);
      return l;
   }
   else {
       l->suiv= rm(l->suiv,n);
       return  l;
   }
}
```