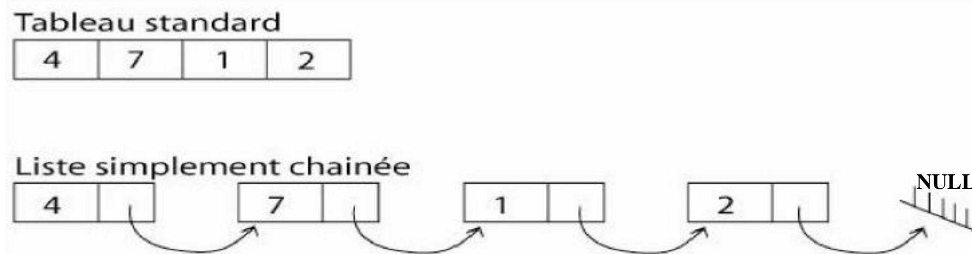
CHAPITRE 3 : LES LISTES CHAÎNÉES

I. Introduction :

Lorsque vous créez un algorithme utilisant des conteneurs, il existe différentes manières de les implémenter. La façon la plus courante, que vous connaissez, est les tableaux. Lorsque vous créez un tableau, les éléments de celui ci sont placés de façon contiguë en mémoire. Pour pouvoir le créer il vous faut connaître sa taille. Si vous voulez supprimer un élément au milieu du tableau, il vous faut recopier les éléments temporairement, réallouer le tableau, puis le remplir à partir de l'élément supprimer. En bref, ce sont beaucoup de manipulations coûteuses en ressources.

Une liste chaînée est différente dans le sens où les éléments de votre liste sont répartis dans la mémoire et reliés entre eux par des pointeurs. Vous pouvez ajouter et enlever des éléments d'une liste chaînée à n'importe quel endroit, à n'importe quel instant, sans devoir recréer la liste entière.



II. La notion de structure autoréférentielle

Une structure *autoréférentielle* (parfois appelée structure récursive) correspond à une structure dont au moins un des champs contient un pointeur vers une structure de même type. De cette façon on crée des éléments (appelés parfois nœuds ou liens) contenant des données, mais, contrairement à un tableau, celles-ci peuvent être éparpillées en mémoire et reliées entre elles par des liens logiques (des pointeurs), c'est-à-dire un ou plusieurs champs dans chaque structure contenant l'adresse d'une ou plusieurs structures de même type.

- Lorsque la structure contient des données et un pointeur vers la structure suivante on parle de *liste chaînée*
- Lorsque la structure contient des données, un pointeur vers la structure suivante, et un pointeur vers la structure précédente on parle de *liste chaînée double*

III. Qu'est-ce qu'une liste chaînée :

III-1. Définition :

Une liste chaînée est une structure comportant des champs contenant des données et un pointeur vers une structure de même type.

III-2. Création d'une structure liste :

```
struct  Nom_de_la_liste
{
          Type var1;

          Type var2;

           struct Nom_de_la_liste * suivant;

};
```

Exemple:

```
struct liste
{

          int val ;
          struct liste *suivant;

};
```

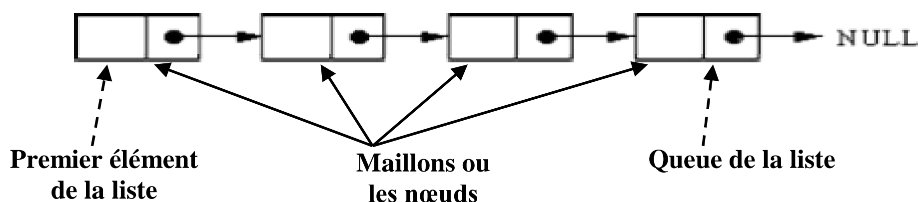
```
        /*   avec typedef   */
typedef struct liste
{
          int val ;
          struct liste *suivant;

}LISTE;
```

III-3. Représentation simpliste d'une liste simplement chaînée :

- *Représentation d'un élément :*

- *Une liste chaînée étant une succession d'éléments, dont le dernier pointe vers une adresse invalide (NULL); voici une représentation possible :*



III-4. Caractéristiques d'une liste simplement chaînée :

- **L'adresse** du premier élément appelée **tête** de la liste, elle permet d'accéder au premier élément.
- Chaque élément est ensuite chaîné au suivant par un pointeur. **Le dernier** élément est associé à un pointeur contenant la valeur NULL, il est appelé **queue** de la liste.

IV. Opérations usuelles sur les listes :

IV-1. Initialisation :

L'initialisation d'une liste est alors triviale : il suffit d'affecter la valeur **NULL** à la variable représentant la tête de la liste.

```
#include<stdio.h>
typedef struct liste
{
        int val;
        struct liste *suivant;

}LISTE;
Int main()
{
LISTE *tete=NULL;
}
```

IV-2. Tester si la liste est vide:

```
int  listevide( LISTE *tete)
{
If(tete ==NULL)
        return (1);
else
        return(0);

}
```

IV-3. Calculer la taille d'une liste:

/*Avec la boucle while*/	/*Avec la boucle for */

int taille_liste (LISTE *tete)
{
int taille=0;
LISTE *p= tete ;
while (p != NULL)
{
p= p->suivant;
taille++;
}
return(taille);
}
	int taille_liste (LISTE *tete)
	{
	int taille=0;
	LISTE *p ;
	for (p= tete ;p != NULL ; p= p->suivant)
	{
	taille++;
	}
	return(taille);
	}

IV-4. Afficher le contenu d'une liste:

```
void  afficher_liste (LISTE  *tete)
{

 LISTE *p ;
  for (p= tete ; p !=NULL ; ,p = p->suivant)
          printf("la valeur de l'element est %d\n",  p->val);
}
```

IV-5. Insertion au début d'une liste :

Passage par valeur pour la tète :

```
LISTE * inserer_debut( LISTE *tete , int v)
{

 LISTE *N;

 N=(LISTE *)malloc(sizeof(LISTE));
N->val=v;
N-> suivant = tete;
tete =N;
return (tete);
}
```

Déclaration globale de la tête :

```
LISTE *tete ;

void  inserer_debut(  int v)
{
 LISTE *N;

 N=(LISTE *)malloc(sizeof(LISTE));
N->val=v;
N-> suivant = tete;
tete =N;
}
```

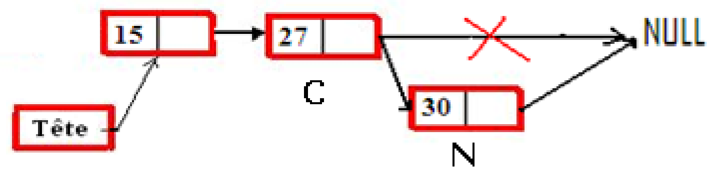
Passage par adresse pour la tète :

```
void  inserer_debut( LISTE **tete , int v)
{

 LISTE *N;

 N=(LISTE*)malloc(sizeof(LISTE));
N->val=v;
N-> suivant =*tete;
*tete =N;
}
```

IV-6. Insertion à la fin d'une liste :



Passage par valeur pour la tête :

```
LISTE * inserer_fin(LISTE *tete,  int v)
{
LISTE *N,*c ;
N=(LISTE *)malloc(sizeof(LISTE)) ;
N->val=v ;
if(tete==NULL)
{
        N-> suivant = NULL;
        tete =N;
}
else
 {
        for(c = tete ; c ->suivant != NULL; c = c ->suivant) ;

        N->suivant=NULL ;
        c -> suivant  =N ;

}
return (tete);
 }
```

Déclaration globale de la tête :

```
LISTE *tete ;

void inserer_fin(int v)
{
LISTE *N,*c ;
N=(LISTE *)malloc(sizeof(LISTE)) ;
N->val=v ;
if(tete==NULL)
{
        N-> suivant = NULL;
        tete =N;
}
else
 {
        for(c = tete ; c ->suivant != NULL; c = c ->suivant) ;

        N->suivant=NULL ;
        c -> suivant  =N ;

}

}
```

Passage par adresse pour la tète :

```
void inserer_fin(LISTE **tete,  int v)
{
LISTE *N,*c ;
N=(LISTE *)malloc(sizeof(LISTE)) ;
N->val=v ;
if(*tete==NULL)
{
        N-> suivant = NULL;
        *tete =N;
}
else
 {
        for(c =*tete ; c ->suivant != NULL; c = c ->suivant) ;

        N->suivant=NULL ;
        c -> suivant  =N ;

}

}
```

IV-7. Supprimer la tête de la liste:

```
LISTE *Supprimer_Debut (LISTE  *tete)
{
        LISTE *c ;
        if(tete !=NULL)
        {
        c= tete ;
        tete = tete ->suivant;
        free(c) ;
        }
return (tete);
}
```

IV-8. Supprimer la queue de la liste:

```
LISTE *Supprimer_Fin (LISTE  *tete)
{
        LISTE *c ;
        if(tete !=NULL)
        {
        if (tete ->suivant  ==NULL)    /*On  a un seul élément : Supprimer la tête de la liste*/
                {c=tete ;
                 tete =NULL ;
                 free(c);
                }
        else
                {
                for(c=tete;c->suivant ->suivant !=NULL;c=c->suivant ) ;

                free(c->suivant );
                c->suivant =NULL ;
                }
        }
return(tete);
}
```