Chapitre IV

LES LISTES CHAINÉES

I. Introduction

Une liste chaînée permet de stocker un ensemble de valeur du même type, comme un tableau. Contrairement au tableau, la taille de la liste chaînée peut varier au cours du temps. En effet, contrairement au tableau, la liste n'est pas allouée en une seule fois, mais chaque élément est alloué indépendamment, sous la forme d'une cellule.

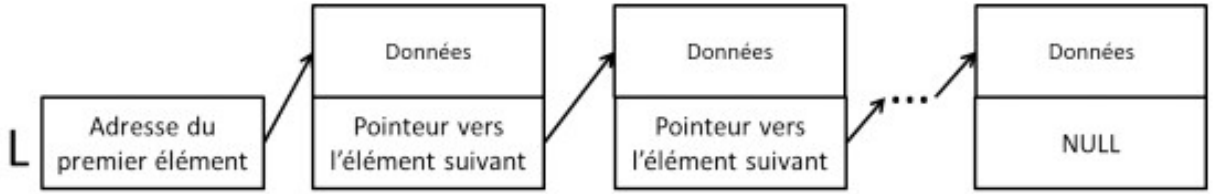
Notion d'élément (cellule)

L'élément de base d'une liste chaînée s'appelle cellule/élément Il est constitué : d'un champ de données et d'un pointeur vers une cellule.

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

Notion de liste chaînée

Dans une liste chaînée, chaque élément pointe, à l'aide d'un pointeur (suivant) vers l'élément suivant dans la liste ; le dernier élément, par définition, n'a pas de suivant, donc son pointeur suivant vaut NULL.



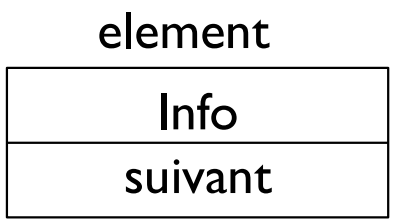
Pour manipuler une liste chaînée, nous manipulons un pointeur sur le premier élément; comme chaque élément « connaît » l'élément suivant, on peut ainsi accéder à tous les éléments de la liste.

Si le pointeur premier vaut NULL, on considérera naturellement que la liste est vide (elle ne contient aucun élément).

Le premier élément de la liste est appelé tête, le dernier élément de la liste est appelé queue.

II. Structure de donnée pour la création d'une liste chaînée

```
struct element
{
        Type info;
        struct element *suivant;
};
```



Exemple

```
struct element
{
        int info;
        struct element *suivant;
};
typedef struct element element;
element *L;
```

III. Principales opérations pour une liste simplement chaînée

1. Création d'une liste vide

```
void initialisation(element *L)
{
    L=NULL;
}
```

2. Affichage des éléments d'une liste

```
void affiche(element *L)
{
element *p;
p=L;
while(p!=NULL)
{
   printf("%d",p→info);
   p=p→suivant;
}
}
```

3. Détermination de la taille d'une liste

```
int taille (element *L)
{ element *p;
  int nb=0;
  p=L;
  while( p!= NULL)
  {  nb=nb + 1;
      p=p→suivant;
   }
   return nb;
}
```

4. Vérification si une liste est vide

```
Int vide (element *L)
{ if (L == NULL)
        return 1;
   else
        return 0;
}
```

5. Ajout d'un élément en tête

Une fois la structure et les pointeurs définis, il est possible d'ajouter un premier élément (cellule) à la liste chaînée, puis de l'affecter au pointeur Tête. Pour cela il est nécessaire :

1. d'allouer la mémoire nécessaire au nouveau élément grâce à la fonction **malloc**.
2. d'assigner au champ "pointeur" du nouveau élément, la valeur du pointeur vers l'élément de tête.
3. définir le nouveau élément comme élément de tête.

```
element* Ajout_tete (element *L, int x)
{ element *p;
   p= (element *)malloc (sizeof (element));
   p→ info= x;
   p→ suivant = L;
   L  = p;
return L;
}
```

6. Ajout d'un élément en queue

L'ajout d'un élément à la fin de la liste chaînée est similaire, à la différence près qu'il faut définir un pointeur (appelé généralement pointeur courant) afin de parcourir la liste jusqu'à atteindre le dernier élément (celui dont le pointeur possède la valeur NULL). Les étapes à suivre sont donc :

1. la définition d'un pointeur courant :
2. le parcours de la liste chaînée jusqu'au dernier nœud :
3. l'allocation de mémoire pour le nouvel élément :
4. faire pointer le pointeur courant vers la nouvelle cellule, et la nouvelle cellule vers NULL :

```
element* Ajout_queue (element *L, int x)

{ element *p, *q;
   p= (element *)malloc (sizeof (element));
   p→ info= x;
   p→ suivant = NULL;

if (L==NULL)
        L= p;
else
 {      q = L;
        while (q →suivant != NULL)
                q= q →suivant;
        q → suivant = p;
   }
     return L;
}
```

7. Ajout d'un élément à une position donnée

Pour l'ajout à une position k donnée, on suppose que la position est valide c'est-à-dire qu'elle est comprise entre 1 et Taille de la liste +1.

```
element*  Ajout_Pos (element *L, int x,int k)
{element *p, *q;
   p= (element *)malloc (sizeof (element));
   p→ info= x;
    if (vide(L)==1 || k==1)
            L=Ajout_tete(L , x);
    else
     { i=1;
       q=L ;
       while (q → suivant != NULL && i<k-1)
        {   q = q → suivant;
            i = i + 1;
        }
       p → suivant = q → suivant;
       q → suivant = p;
      }
  return L;
   }
```

P->suivant=L;
L=p;

8. Suppression de l'élément en tête

```
element* Supp_tete( element *L)
{ element *p;
   if (vide( L)==0)
   {   p=L;
       L = p→suivant;
       free(p);
    }
return L;
}
```

9. Suppression de l'élément en queue

```
Element* Supp_queue( element *L)
{ element *p,*q;
  if (vide(L)==0)
      if ( L →suivant == NULL)
           L=Supp_tete(L);
      else
      { p = L ;
        while ( (p →suivant) →suivant != NULL)
             p=p →suivant;
        q = p →suivant;
        p →suivant= NULL;
        free(q);
      }
     return L;
}
```

Activité I

Soit la définition de la structure suivante :

```
Struct produit
{
int numero;//numéro d'un produit
char nom[20];//nom du produit
};
```

1. Ecrire la fonction Remplissage(...) permettant de remplir une liste simplement chainée de N produits.

2. Ecrire la fonction AFFICHE(...) qui affiche les éléments de la liste.

```
struct element

{ struct produit p;

struct element *suivant;

};

typedef struct element element;

element*L;
```

```
element* remplissage(element*L, int N)
{element *q;
Struct produit t;
int i;
For(i=1;i<=N;i++)
{q=(element*)malloc(sizeof(element));
   printf(« donner le numero et le nom du produit %d: »,i);
   scanf(« %d%s »,&q->p.numero,q->p.nom);
    scanf(« %d%s »,&t.numero,t.nom);
   q->suivant=L;    L=Ajout_tete(L,t);
   L=q;
}
Return L;
}
```

```
Void affiche(element *L)
{element *k;
k=L;
While(k!=NULL)
 {   printf(« le numéro= %d et le
nom=%s »,k->p.numero,k->p.nom);
 k=k->suivant;
}
}
```

```
void main()
{ element *L1;
int N;
Do
{ printf(« donner le nombre des éléments de la liste »);
scanf( "%d ",&N);
}while(N<=0);
L1=NULL;
Remplissage(L1,N);
Affiche(L1);
}
```

Activité 2

Écrire une fonction qui permet de fusionner deux listes d'entiers afin d'avoir une troisième liste chaînée triée.