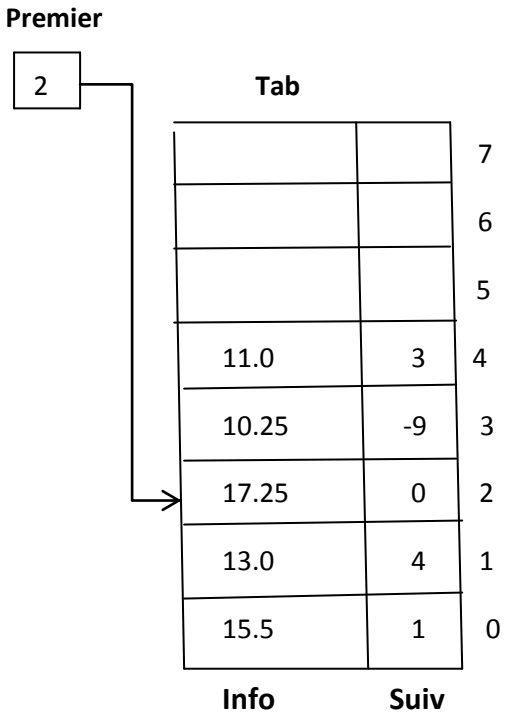
Les listes

INTRODUCTION

La liste est une structure de données qu'on peut étudier sous deux formes : Contigu ou chaînée cela veut dire sous forme d'un tableau (c'est une simulation) ou bien sous forme d'une suite de nœuds reliés entre eux par des liens de chaînage qu'on appelle par abus de langage « adresses » ou « pointeurs ».

Les listes sous forme de tableau



Comme on le constate sur ce design la liste est composée d'un tableau appelé **Tab** et un indice **Premier** pas de parcours mais il indique seulement le premier de la liste L.

- Premier est un entier.
- Tab est un vecteur où chaque case est constituée de deux champs : **Info** et **suiv**. le premier nommé contient l'information et le second le N° de la case de l'information suivante.
- **Maxliste** est une constante pour indiquer la taille du tableau. Dans cet exemple Maxliste = 8.

Si on parcoure cette liste L, on aura : 17.25 15.5 13.0 11.0 et 10.25.

Dans notre exemple Premier = 2, cela veut dire que le premier élément de la liste L se trouve dans la case N° 3 à partir du bas (case 0, case 1, case 2).

```
/* Implementation de la liste de l'exemple*/

#define Maxliste  8

typedef struct item  item;

struct  item  {

                float          info ;

                int    suiv ;

               } ;

typedef struct Liste  Liste ;

struct Liste    {

                int Premier ;

                item Tab[Maxliste] ;

               } ;

Liste  L ;
```

Par convention quand la liste est vide, l'indice premier est égal à -9 ; et si une information n'a pas de suivant, alors suiv = -9.

Elle est pleine si aucune case du tableau n'est libre, c'est-à-dire que le nombre d'informations dans la liste est égal à Maxliste.

Essayons d'écrire quelques opérations de base concernant cette liste L.

```
/* Initialisation*/

Void   InitialiserListe(Liste *L)

{

        L.Premier = -9 ;

}
```

```
/* Liste vide oui = 1 non = 0 */
Int ListeVide(liste L)
{
        If L.Premier = -9
            Return 1 ;
        Return 0 ;
}
```

Avant d'écrire la fonction ListePleine, écrivons une procédure qui parcoure une telle liste.

```
/* Parcours de la  liste*/
Void  parcoursListe(Liste  L)
{
   Int i = L.Premier ;
   While i != -9
        i = L.Tab[ i ].suiv ;
}
```

```
/* Liste pleine  oui = 1 non = 0*/
Int ListePleine(Liste  L)
{
   Int Co = 0 ;
   Int i = L.Premier ;
   While i != -9
     {
          Co = Co+1;
          i = L.Tab[ i ].suiv ;
      }
    If Co = MaxListe     /* Toutes les cases sont occupées */
       Return 1;
     Return 0;
}
```

Pour ajouter ou supprimer une information (note+ suivant) dans cette liste sous forme de tableau demande beaucoup de précautions. C'est-à-dire que sa gestion n'est pas aisée.

La liste dynamique chaînée est très avantageuse !

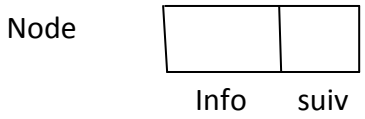
Etudions-la.

Les listes dynamiques chaînées.

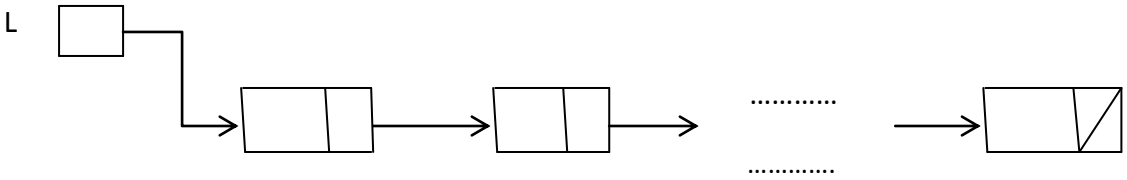
Une liste chaînée est composée d'éléments qu'on appelle nœuds ou maillons ou chaînons ou nodes ou simplement éléments. Allons-y pour « **node** ».

Un node est une structure (enregistrement) composé de deux champs : L'un pour contenir une information et l'autre pour contenir l'adresse de son suivant. Si le node n'as pas de suivant alors cette adresse est égale à « nulle part ailleurs » disons simplement NULL.

Représentation schématique d'un noeud node.

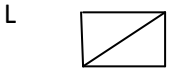


Représentation schématique d'une liste L.

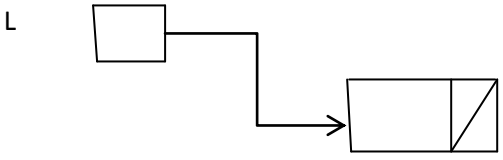


On remarque que le dernier nœud qui n'a pas de suivant, il y a un slash dans le champ suiv c'est-à-dire NULL.

Une liste L vide sera représentée par :



Une liste L dite « singleton » est celle qui contient un seul nœud.



```
/* Implémentation*/

#include <stdlib.h>

Typedef struct   node   node;

Struct  node {

                Item    info;

                Struct  node    *suiv;

               };

Typedef   node*        liste;


Liste  L = NULL ;          /* Déclaration de la liste L et son initialisation*/
```

L'écriture de quelques opérations de base, de cette liste en langage, seront sur le fascicule intitulé « Les listes en C ».