Programmation et Structure de Données

Structure de données de Base
Les listes chainées partie 1

ENS KATEB

Listes chainées : Définition

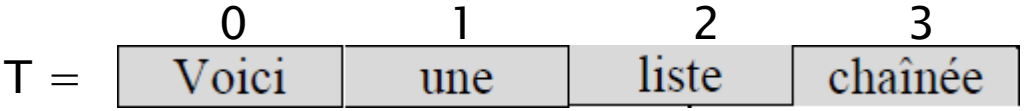
1. Définition :

Une liste est un tableau d'éléments de même type où :

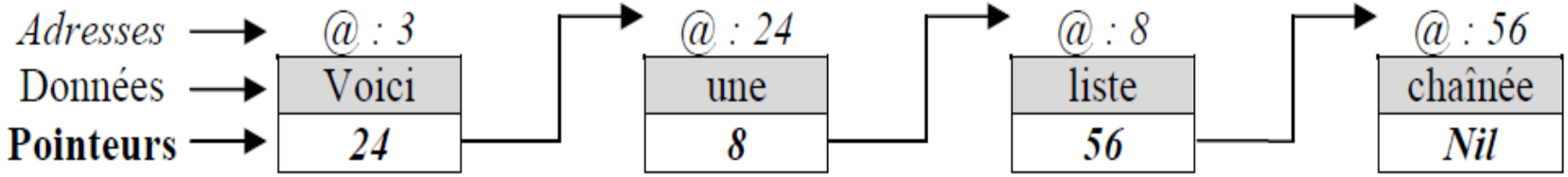
1. La dimension est variable (peut changer au cour de l'exécution du programme)
2. Les éléments ne sont pas contigus (chaque élément est connecté au suivant à l'aide d'un pointeur)
3. L'accès aux éléments de la liste est effectué uniquement par sa tête c'est à dire son premier élément.

Exemple :

Un Tableau de mots :



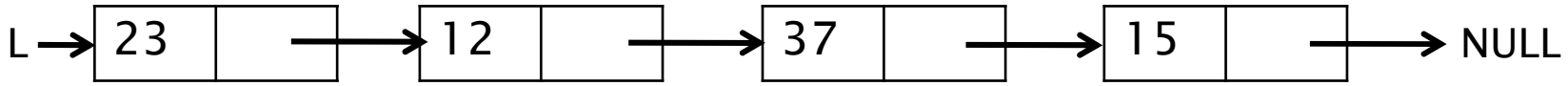
Une liste chainée de mots :



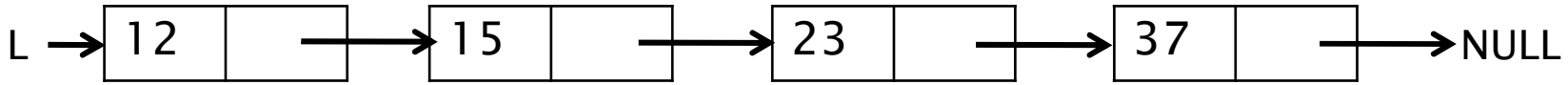
Listes chainées : types

2. Différents types des listes chaînées:

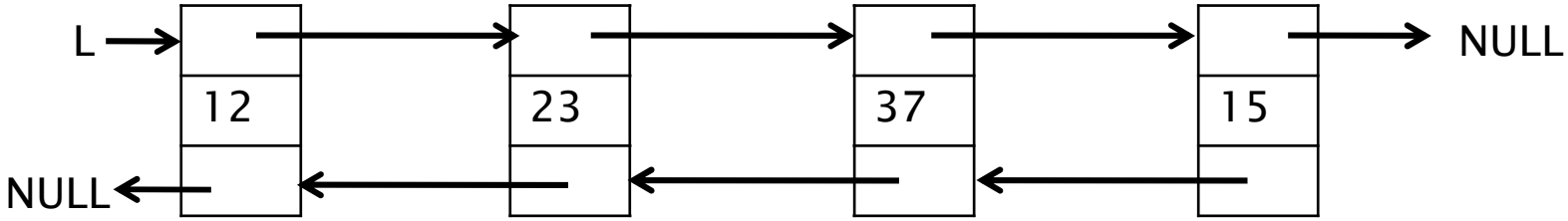
- Liste simplement chaînée :



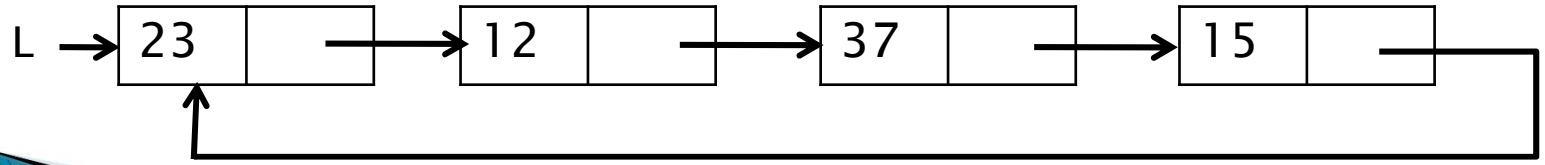
- Liste simplement chaînée et ordonnée :



- Liste doublement chaînée :



- Liste circulaire :



Listes chainées : Déclaration

3. Déclaration et manipulation des listes chainées :

Définir un type pointeur vers élément:

elem : Struct
Début
    Val : entier;
    *suiv : elem;
Fin;

Déclaration d'une variable L pointeur vers elem

Var
***L : elem;** donc L est une adresse qui pointe vers « elem » **c'est un objet dynamique**

Initialisation par vide :

L ← NULL; NULL représente l'adresse vide

Déclarer un élément, lui affecter une valeur puis le lier à la liste L :

Var
*E : elem; **objet dynamique qui représente l'adresse d'un élément**
E = nouveau (elem); **(en C++ E = new (elem);)** pour créer un nouveau objet elem
E -> val ← 34; E -> suiv ← NULL;
L ← E; À ce niveau notre liste L contient un élément

Pour accéder aux attributs d'un objet dynamique : **on utilise l'opérateur « -> » au lieu de l'opérateur « . »**

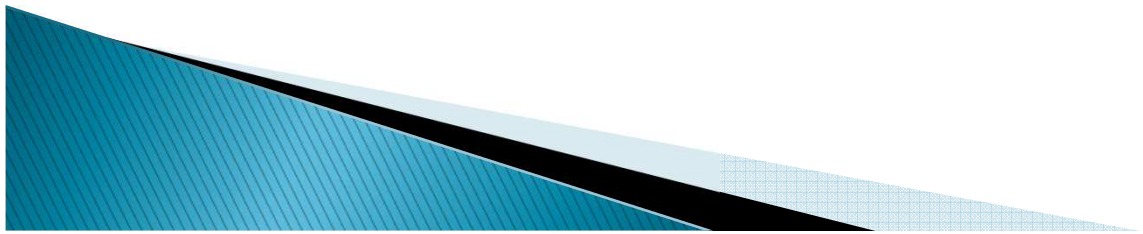
Listes chainées : Déclaration

En C++ :

```
struct elem
{
    int val;
    elem *suiv;
};
elem *L=NULL;
elem *E= new(elem); E -> val = 34;  E -> suiv = NULL;
L = E;
cout << "la valeur du 1ere element de la liste : ";
cout << L->val;
}
```

4. Trois (03) principales opérations :

1. Parcours de la liste
2. Ajout d'un élément
3. Suppression d'un élément



Listes chainées : Déclaration

Dans ce qui suit on va utiliser ces deux déclarations suivantes pour traiter les listes simplement chainées : **Structure 1**
et pour traiter les listes doublement chainées : **Structure 2**

Structure 1

```
struct elem_s
{
 int D;
 elem_s * suiv;
};
```

Structure 2

```
struct elem_d
{
 int D;
 elem_d * suiv;
 elem_d * prec;
};
```

Listes chainées : Parcours

1. **Cas d'une liste simple :**

Le parcours permet, par exemple, de lire ou d'afficher les éléments de la liste

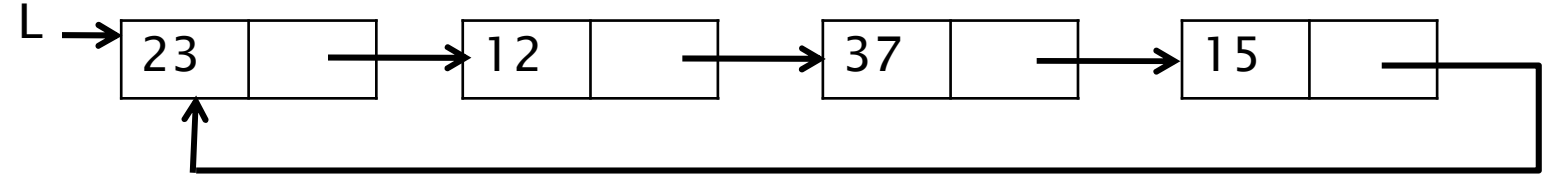
Ex : écrire un sous programme qui prend en argument une liste et affiche les valeurs des éléments de cette liste

```
void aff_s (elem_s *L)
{
    if(L!=NULL)
    {
      elem_s *T = L;
      cout <<"Voici les elements de votre Liste Simple : \n";
      while(T!=NULL)
       {
            cout << T->D <<"   ";
            T = T->suiv;
       }
    }
}
```

(Travail à faire : même exemple avec la récursivité)

Listes chainées : Parcours

2. Cas d'une liste circulaire :




```
void aff_cir (elem_s *L)
{
    if(L!=NULL)
    {
        elem_s *T = L;
        cout <<"Voici les elements de votre Liste Circulaire : \n";
        do
         {
            cout << T->D <<"   ";   T = T->suiv;
         }
        while(T != L)
     }
}
```

Listes chainées : Insertion

Dans une liste on peut ajouter un élément dans trois positions différentes : en tête (début) de liste, en queue (Fin) de Liste et en Milieu de liste dans une position précise par exemple.

Cas d'une liste simple :
A – En tête (début) de liste : (on va utiliser un s/prg)

```
elem_s * ajoutDeb_s (elem_s * L)
{
      elem_s *E = new(elem_s);
      cout <<"Donner la valeur du nouvel element : ";
      cin >> E->D;
      E->suiv = L;
      return E;
 }
```

Listes chainées : Insertion

Cas d'une liste simple :
B – En queue (Fin) de liste :

```
elem_s * ajoutFin_s (elem_s *L)
{
        elem_s *E = new(elem_s);
        cout <<"Donner la valeur du nouvel element : "; cin >> E->D;
        E->suiv = NULL;
        if(L==NULL) return E;
        else
           {
             elem_s *T = L;
             while (T->suiv != NULL) T = T->suiv;
              T->suiv = E;  return L;
            }
}
```