

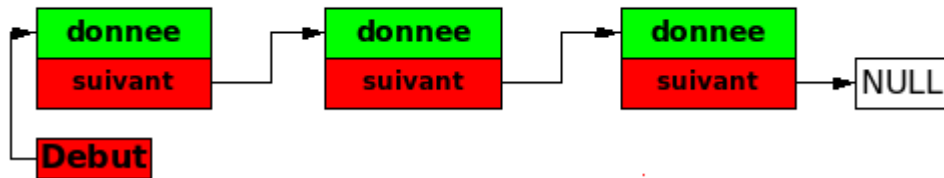
Chapitre 4 : LISTES CHAÎNÉES

I- LISTES SIMPLEMENT CHAÎNÉES

1- Définition

Les listes chaînées sont des structures de données semblables aux tableaux sauf que l'accès à un élément ne se fait pas par *index* mais à l'aide d'un *pointeur*.

Liste simplement chaînée



La liaison entre les éléments se fait grâce à un pointeur.

Le pointeur **suivant** du dernier élément doit pointer vers **NULL** (la fin de la liste).

Pour accéder à un élément, la liste est parcourue en commençant avec la tête, le pointeur suivant permettant le déplacement vers le prochain élément.

Le déplacement se fait dans une seule direction, du premier vers le dernier élément.

2- La construction du prototype d'un élément de la liste

Pour définir un élément de la liste le type **struct** sera utilisé.

L'élément de la liste contiendra un champ **donnee** et un pointeur suivant.

Le pointeur **suivant** doit être du même type que l'élément, sinon il ne pourra pas pointer vers l'élément.

Le pointeur "**suivant**" permettra l'accès vers le prochain élément.

```
typedef struct ElementListe {  
    char *donnee;  
    struct ElementListe *suivant;  
}Element;
```

Remarque :

Pour avoir le contrôle de la liste il est préférable de sauvegarder Le premier élément.

Le pointeur **debut** contiendra l'adresse du premier élément de la liste.

3- Opérations sur les listes chaînées

Initialisation

Prototype de la fonction :

```
void initialisation (Liste *debut);
```

Cette opération doit être faite avant toute autre opération sur la liste.

Elle initialise le pointeur **debut** avec le pointeur **NULL**,

La fonction

```
void initialisation (Listes *debut){
    debut = NULL;
}
}
```

4- Insertion d'un élément dans la liste

Voici l'algorithme d'insertion et de sauvegarde des éléments :

- Déclaration d'élément(s) à insérer
- Allocation de la mémoire pour le nouvel élément
- Remplir le contenu du champ de données
- Mettre à jour les pointeurs vers le 1er et le dernier élément si nécessaire.

Pour ajouter un élément dans la liste il y a plusieurs situations :

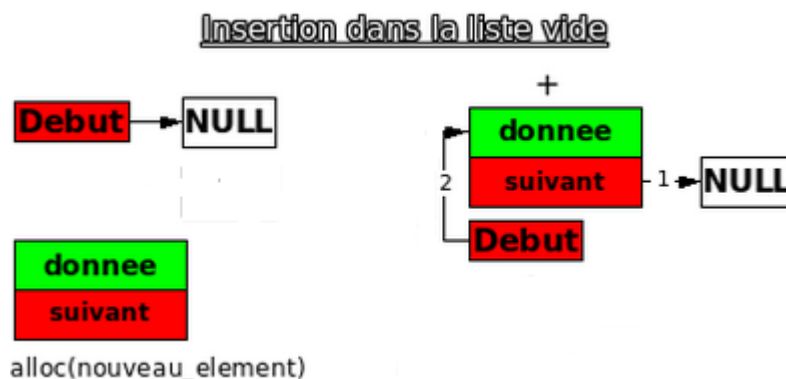
1. Insertion dans une liste vide
2. Insertion au début de la liste
3. Insertion à la fin de la liste
4. Insertion ailleurs dans la liste

a- Insertion dans une liste vide

Prototype de la fonction :

```
void ins_dans_liste_vide (Liste *debut, char *donne);
```

Étapes :



La fonction

/* insertion dans une liste vide */

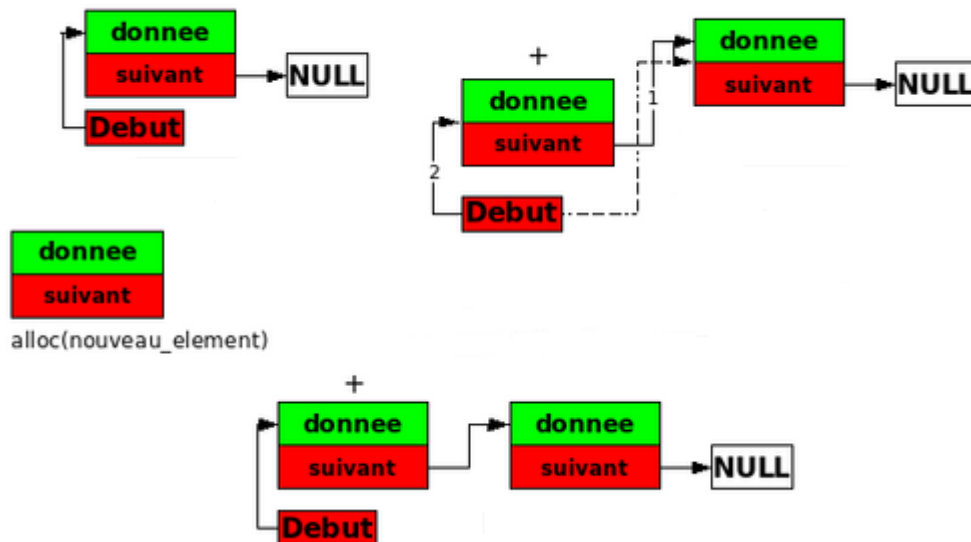
```
void ins_dans_liste_vide (Listes * debut, char * val){
    Listes *NE;
    NE = (Listes *) malloc (sizeof (Listes)) ;
    strcpy (NE->donnee, val);
    NE->suivant = NULL;
    debut = NE;
}
}
```

b- Insertion au début de la liste

Prototype de la fonction :

```
void ins_debut_liste (Liste *liste, char *val);
```

Insertion au début de la liste



La fonction

/* insertion au début de la liste */

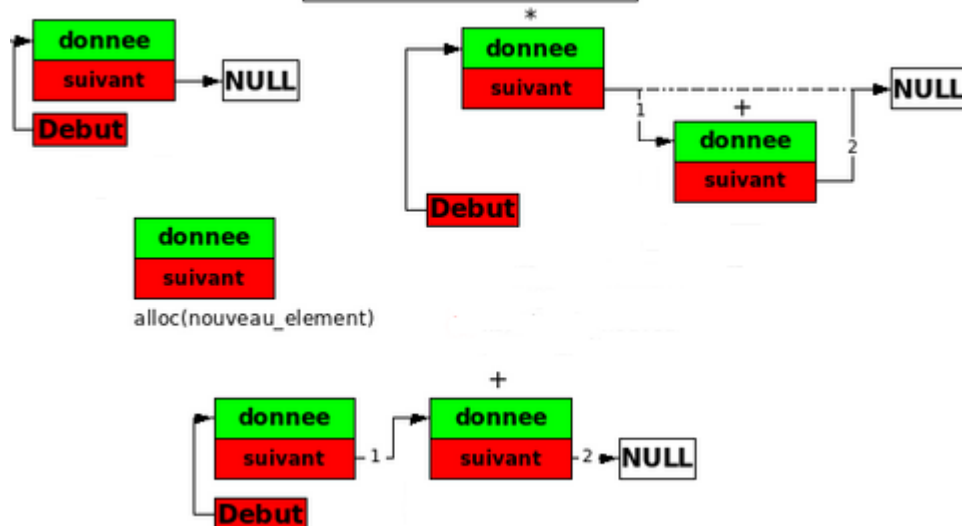
```
void ins_debut_liste (Listes * liste, char *val){
    Listes *NE;
    NE = (Listes *) malloc (sizeof (Listes))
    strcpy (NE ->donnee, val);
    NE->suivant = debut;
    debut = NE;
}
```

c- Insertion à la fin de la liste

Prototype de la fonction :

void ins_fin_liste (Liste *liste, Element *courant, char *donne);

Insertion à la fin de la liste



La fonction

/*insertion à la fin de la liste */

```
void ins_fin_liste (Liste * liste, char *val){
    Listes *NE;
    Listes * courant ;
    NE = (Listes *) malloc (sizeof (Element)) ;
    strcpy (nouveau_element->donnee, val);
    for(courant=debut ;courant->suivant !=NULL ;courant=courant->suivant)
    ;
}
```

```

courant->suivant = NE;
NE ->suivant = NULL;
}

```

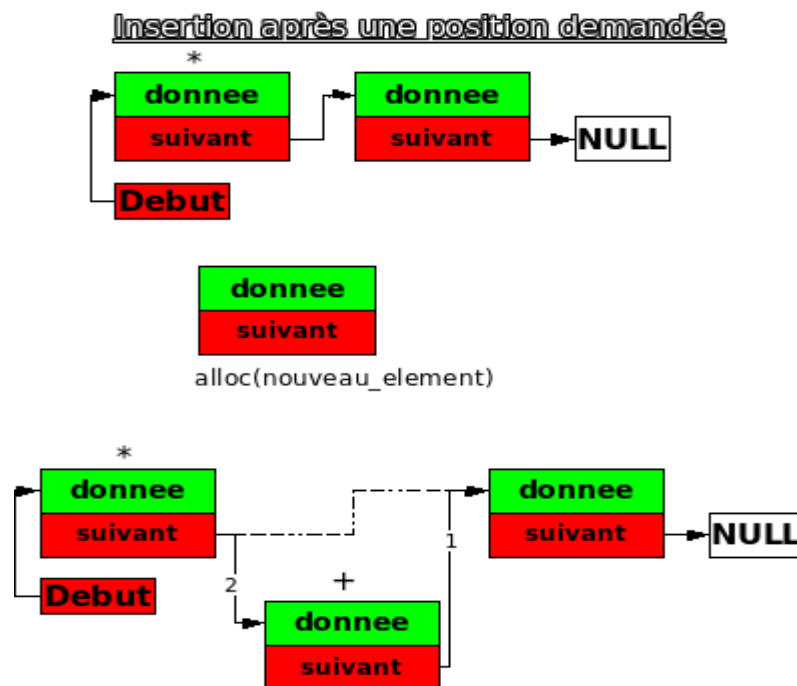
d- Insertion ailleurs dans la liste

Prototype de la fonction :

void ins_liste (Liste *liste, char *val, int pos);

L'insertion s'effectuera après une certaine position passée en argument à la fonction.

Si la position indiquée ne doit pas être le dernier élément. Dans ce cas il faut utiliser la fonction d'insertion à la fin de la liste.



La fonction

```

/* insertion à la position demandée */
int ins_liste (Liste * liste, char *val, int pos){
    if (pos < 1 || pos >= nb)
        return -1;
    Listes *courant;
    Listes *NE;
    int i;
    NE = (Listes *) malloc (sizeof (Listes));
    courant =debut;
    for (i = 1; i < pos; ++i)
        courant = courant->suivant;
    strcpy (NE->donnee, val);
    NE ->suivant = courant->suivant;
    courant->suivant = NE;
    return 0;
}

```

5-Suppression d'un élément dans la liste

Voici l'algorithme de suppression d'un élément de la liste :

- utilisation d'un pointeur temporaire pour sauvegarder l'adresse d'éléments à supprimer.
- l'élément à supprimer se trouve après l'élément courant

Faire pointer le pointeur suivant de l'élément courant vers l'adresse du pointeur suivant de l'élément à supprimer

- libérer la mémoire occupée par l'élément supprimé

Pour supprimer un élément dans la liste il y a plusieurs situations :

- 1. Suppression au début de la liste
- 2. Suppression ailleurs dans la liste

a- Suppression au début de la liste

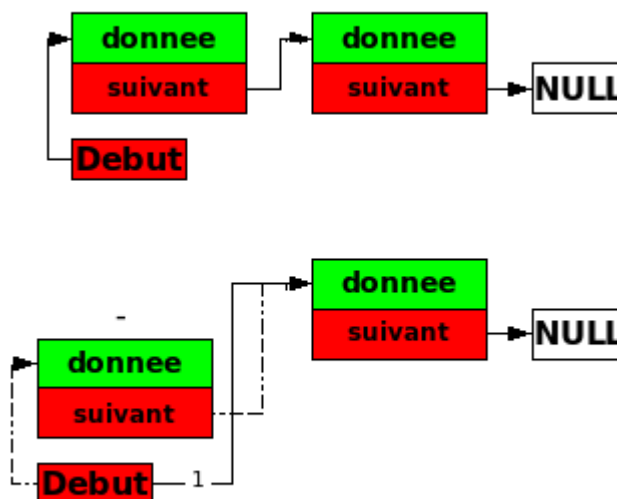
Prototype de la fonction:

void supp_debut (Liste *liste);

Étapes:

- le pointeur supp_elem contiendra l'adresse du 1er élément
- le pointeur debut pointera vers le 2ème élément
- la taille de la liste sera décrémentée d'un élément

Suppression au début de la liste



```

supp_element = liste->debut;
(1) liste->debut = liste->debut->suivant;

```

La fonction

```

/* suppression au début de la liste */
void supp_debut (Listes * liste){
    Element *supp_element;
    supp_element = liste->debut;
    liste->debut = liste->debut->suivant;
    free (supp_element->donnee);
    free (supp_element);
}

```

b- Suppression ailleurs dans la liste

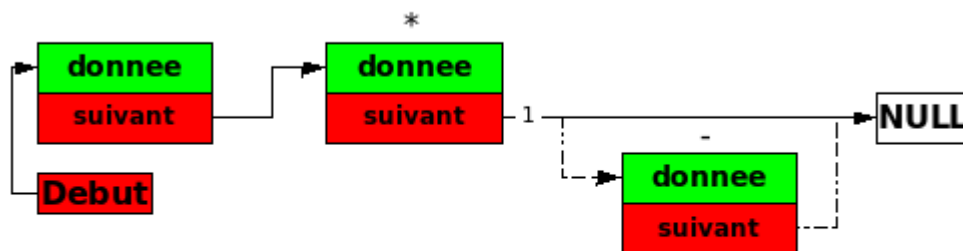
Prototype de la fonction:

void supp_dans_liste (Liste *liste, int pos);

Étapes:

- le pointeur `supp_elem` contiendra l'adresse vers laquelle pointe le pointeur suivant d'élément courant
- le pointeur suivant de l'élément courant pointera vers l'élément sur lequel pointe le pointeur suivant de l'élément qui suit l'élément courant dans la liste

Suppression après l'avant dernier élément



```
supp_element = courant->suivant;
(1) courant->suivant = courant->suivant->suivant;
```

La fonction

/* supprimer un element après la position demandée */

```
void supp_dans_liste (Liste * liste, int pos){
    int i;
    Element *courant;
    Element *supp_element;
    courant = liste->debut;
    for (i = 1; i < pos; ++i)
        courant = courant->suivant;
    supp_element = courant->suivant;
    courant->suivant = courant->suivant->suivant;
    free (supp_element->donnee);
    free (supp_element);
}
```

5- Affichage de la liste

Pour afficher la liste entière il faut se positionner au début de la liste (le pointeur `debut` le permettra).

Ensuite en utilisant le pointeur `suivant` de chaque élément la liste est parcourue du 1er vers le dernier élément.

La condition d'arrêt est donnée par le pointeur `suivant` du dernier élément qui vaut `NULL`.

La fonction

```
/* affichage de la liste */
void affiche (Liste * liste){
    Element *courant;
    courant = liste->debut;
    while (courant != NULL){
        printf ("%p - %s\n", courant, courant->donnee);
        courant = courant->suivant;
    }
}
```