

# TP - Structures de données "LSC"

## Introduction

Dans cet exercice, nous abordons plusieurs aspects des listes chaînées, notamment l'insertion de nœuds au début et à la fin de la liste, la division de la liste en groupes en fonction des critères spécifiques, et le tri des éléments selon un ordre particulier. Chaque fonction implémentée, de l'insertion à la libération de mémoire en passant par le tri, illustre un aspect essentiel de la manipulation des listes chaînées en

EX:

code source : [github.com/youssef-hachimi](https://github.com/youssef-hachimi)

### 1- Insérer un nœud au début de la liste:

1. Définition de la structure du nœud :
2. Définition de la fonction `inserer\_debut` :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  // Définition de la structure de nœud
6  typedef struct Liste_Chaine
7  {
8      char nom[20];
9      char prenom[20];
10     float note;
11     struct Liste_Chaine *suivant;
12 } LSC;
13
14 // les variables globales
15 LSC *Debut = NULL;
16 LSC *lsup = NULL;
17 LSC *linf = NULL;
18
19 //nouveau nœud
20 LSC *Nouveau_Noeud(char nom[20], char prenom[20], float note)
21 {
22     LSC *nv = (LSC *)malloc(sizeof(LSC));
23     if (nv != NULL)
24     {
25         strcpy(nv->nom, nom);
26         strcpy(nv->prenom, prenom);
27         nv->note = note;
28         nv->suivant = NULL;
29     }
30     return nv;
31 }
32
33 // insérer un nœud au début
34 void inserer_debut(char nom[20], char prenom[20], float note)
35 {
36     LSC *nv = Nouveau_Noeud(nom, prenom, note);
37     if (nv != NULL)
38     {
39         nv->suivant = Debut;
40         Debut = nv;
41     }
42 }
```


## 2 - Insérer un nœud à la fin de la liste :

### - 1. Définition de la fonction `inserer\_fin` :

```
1  // insérer un nœud à la fin
2  void inserer_fin(char nom[20], char prenom[20], float note)
3  {
4      LSC *aide, *nv = Nouveau_Noeud(nom, prenom, note);
5      aide = Debut;
6      if (nv != NULL)
7      {
8          if (Debut == NULL)
9          {
10             Debut = nv;
11          }
12          else
13          {
14             while (aide->suivant != NULL)
15             {
16                 aide = aide->suivant;
17             }
18             aide->suivant = nv;
19          }
20      }
21  }
22
```

### 3- Libérer toute la mémoire utilisée per la liste chaînée :

- 1. Définition de la fonction `liberer\_memoire` :



```
1
2 // libérer la mémoire
3 void liberer_memoire()
4 {
5     LSC *temp;
6     while (Debut != NULL)
7     {
8         temp = Debut;
9         Debut = Debut->suivant;
10        free(temp);
11    }
12    lsup = NULL;
13    linf = NULL;
14 }
```

#### 4- Diviser la classe en deux groupes :

##### 1. Définition de la fonction `diviser\_classe` :

```
1 // diviser la classe en deux groupes
2 void diviser_classe()
3 {
4     LSC *aide1, *x;
5     aide1 = Debut;
6     while (aide1 != NULL)
7     {
8         x = Nouveau_Noeud(aide1->nom, aide1->prenom, aide1->note);
9         if (x != NULL)
10        {
11            if (aide1->note >= 10)
12            {
13                if (lsup == NULL)
14                    lsup = x;
15            }
16            else
17            {
18                if (linf == NULL)
19                    linf = x;
20            }
21        }
22        aide1 = aide1->suivant;
23    }
24 }
25
```

## 5- Afficher les étudiants de chaque groupe (version itérative) :

Définition de la fonction `afficher\_groupes\_iteratif` :

```
1 // afficher les étudiants de chaque groupe
2 void afficher_groupes_iteratif(LSC *liste)
3 {
4     LSC *temp = liste;
5     while (temp != NULL)
6     {
7         printf("%s %s %.2f\n", temp->nom, temp->prenom, temp->note);
8         temp = temp->suivant;
9     }
10 }
11
```

## 6- Trier les étudiants:

```
1
2 // trier les étudiants par note
3 void trier(LSC *L)
4 {
5     LSC *i, *j, *min;
6     float aux1;
7     char aux2[20];
8     for (i = L; i->suivant != NULL; i = i->suivant)
9     {
10         min = i;
11         for (j = i->suivant; j != NULL; j = j->suivant)
12         {
13             if (j->note < min->note)
14                 min = j;
15         }
16         // Échanger les données des nœuds si nécessaire
17         if (min != i)
18         {
19             aux1 = i->note;
20             i->note = min->note;
21             min->note = aux1;
22             strcpy(aux2, i->nom);
23             strcpy(i->nom, min->nom);
24             strcpy(min->nom, aux2);
25             strcpy(aux2, i->prenom);
26             strcpy(i->prenom, min->prenom);
27             strcpy(min->prenom, aux2);
28         }
29     }
30 }
31
```

## Fonction main:



```
1  int main()
2  {
3      // pour tester
4      inserer_debut("youssef","hachimi",16);
5      inserer_debut("yahya","mohib",14);
6      inserer_debut("hamza","mouanid",18);
7      inserer_debut("ayoub","lchehb",8);
8
9      // Affichage de la liste initiale
10     printf("Liste initiale :\n");
11     afficher_groupes_iteratif(Debut);
12     printf("\n");
13
14     // Division de la classe en deux groupes
15     diviser_classe();
16
17     // Affichage des deux groupes
18     printf("Liste des etudiants admis :\n");
19     afficher_groupes_iteratif(lsup);
20     printf("\n");
21
22     printf("Liste des etudiants non admis :\n");
23     afficher_groupes_iteratif(linf);
24     printf("\n");
25
26     // Trier les étudiants admis par ordre de mérite (note)
27     trier(lsup);
28
29     // Affichage de la liste des étudiants admis triée
30     printf("Liste des etudiants admis triee par ordre de merite :\n");
31     afficher_groupes_iteratif(lsup);
32     printf("\n");
33
34     // Libération de la mémoire utilisée par la liste chaînée
35     liberer_memoire();
36
37     return 0;
38 }
```