TP1

LISTES SIMPLEMENT CHAINEES

ENSEIGNANTE : MME ONS BEN ROMDHANE — **SEMESTRE** : 2 - 2023/2024

MATIERE : ALGORITHMIQUE AVANCE — **GROUPES** : 3EME INFO H, I & J

Objectifs

Au terme de cet atelier, vous saurez :

Implémenter les primitives fondamentales sur les listes simplement chaînées.

Résoudre des problèmes avec des listes simplement chaînées.

Durée estimative

3h

FUSION DE DEUX LISTES

Soit la partie déclarative générale suivante :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

struct cellule
{
   int info;
   cellule * suiv;    // Pointeur sur la cellule suivante dans la liste chaînée
};

typedef cellule * liste;    //renommer le type cellule * par liste
```

Les modules lacunaires présentés ci-dessous permettent de fusionner deux listes chaînées *L1* et *L2* triées dans l'ordre croissant (selon le champ *info*), en une troisième liste chaînée *L3* triée aussi dans le même ordre.

Travail demandé

1) Vous êtes appelés à remplacer les lettres alphabétiques [A]...[AB] par un code C (une déclaration, une expression ou une instruction).

2) Développer les fonctions *remplir*() et *affiche().*

```
liste allo_cell()
{  /* Cette fonction permet d'allouer dynamiquement une structure de type struct cellule
      et de retourner son adresse */

    [A]
    nouv = (liste) malloc (sizeof(struct cellule));
    if (!nouv)
       printf("Allocation impossible");
    else
       [B]
}
```

// **NB :** Utiliser la fonction *allo_cell()* dans les fonctions suivantes.

```
[C]  // l'entête de la fonction copie
{
 /* Cette fonction permet de copier tous les éléments d'une liste chaînée (L est l'adresse de
 son premier élément) dans une deuxième liste chaînée en respectant le même ordre. La
 fonction retourne l'adresse du premier élément de la deuxième liste. */

 //p est un pointeur de parcours de la liste chaînée L
 [D]

 // Pointeur sur un nouvel élément de la nouvelle liste en cours de création
 liste nouv;

 // Pointeur sur le dernier élément de la nouvelle liste en cours de création
 Liste pfin;

// Pointeur sur le premier  élément de la nouvelle liste en cours de création
 liste Lres = NULL;

 while(p != NULL)
 {  [E]       // Allocation dynamique d'une structure element.
    nouv->info = p->info;
    nouv->suiv = NULL;

    if([F])    // La deuxième liste est encore vide
    { Lres = nouv;
      pfin = nouv;
    }
    else
    { [G]
      [H]
    }
```

```
    p = p->suiv;
  }
 [I]
}
```

```
liste inserer_tete(liste L, int val)
{
```

/* Permet d'insérer l'entier *val* au début d'une liste chaînée (non vide) dont la première cellule est pointée par *L*. */

```
  liste nouv;
  [E]
  nouv->info = val;
  [J]
  [K]
  return L;
}
```

```
[L] inserer_fin(liste queue, int val)
{
```

/* Permet d'insérer l'entier *val* à la fin d'une liste chaînée (supposée non vide) dont la dernière cellule est pointée par *queue*.*/

```
  liste nouv;
  [E]
  nouv->info = val;
  [M]
  [N]
}
```

```
void inserer_milieu(liste adr, int val)
{
```

/* Permet d'insérer l'entier *val* au milieu d'une liste chaînée (supposée non vide) après La cellule pointée par *adr*. */

```
  liste nouv;
  [E]
  nouv->info = val;
  [O]
  [P]
}
```

```
liste recherche_pos(liste L, int val)
{
```

/* Cette fonction permet de chercher la position d'insertion de l'entier val dans une liste chaînée, dont la première cellule est pointée par *L,* de telle manière que cette liste reste toujours triée dans un ordre croissant. On distingue **trois cas** :

a) Si l'entier *val* est inférieur au premier élément de la liste chaînée, la fonction retourne NULL.

b) Si on parcourt la liste chaînée à partir du début et on trouve un élément qui est supérieur ou égal à *val*, la fonction retourne l'adresse du prédécesseur (précédent) de l'élément trouvé.

c) Sinon, la fonction retourne l'adresse du dernier élément de la liste chaînée. */

```
  liste p = L;        // Pointeur de parcours de la liste chaînée L.
  liste prec = NULL;        // Résultat de la fonction.
  int stop = 0; //<=> Faux


  while([Q])
  { if (p->info < val)
    {  [R]
       [S]
    }
    else
       [T]
  }
  return prec;
}
```

```
liste fusion (liste L1, liste L2)
{
```

/* Permet de fusionner deux listes chaînées *L1* et *L2* triées dans l'ordre croissant, selon le champ *info*, en une troisième liste chaînée *L3* triée aussi dans le même ordre et qui sera retournée par la fonction fusion.

On distingue trois cas :

a) Si *L1* est vide alors copier les éléments de *L2* dans *L3*.

b) Si *L2* est vide alors copier les éléments de *L1* dans *L3*.

c) Si *L1* et *L2* sont non vides alors copier d'abord tous les éléments de *L1* dans *L3*. Puis, insérer chaque élément de *L2* à sa bonne position dans *L3*, de telle sorte que cette dernière reste triée.

Pour trouver cette position d'insertion, on procède comme suit : pour chaque élément de *L2*, on parcourt *L3* à partir du début jusqu'à trouver un élément qui lui est supérieur ou

égal, dans ce cas l'insertion se fait avant l'élément trouvé. Sinon, l'insertion se fait à la fin de *L3*. */

```
liste L3 =NULL;   // La liste chaînée finale après la fusion
liste p2 = L2;    // Pointeur de parcours de la liste chaînée L2
liste prec;

if ([U])
   return L2;
else
{ if([V])
   return L1;
 else
 { // Copie de L1 dans L3
   L3 = copie(L1);

   // Insertion des éléments de L2 dans L3
   while([W])
   {  prec = [X]  /* Recherche de la position d'insertion de l'élément courant
                     de L2  dans la liste L3.*/
      if(prec == NULL)
          [Y]
      else
      { if([Z])
          [AA]

        else
          [AB]
      }
      p2 = p2->suiv;
   }
   return L3;
  }
 }
}
```

```
int main()
{liste L1, L2, L3;  //L1, L2 et L3 sont trois listes simplement chaînée

 /* La fonction remplir() permet de créer une liste simplement chaînée triée dans l'ordre
 croissant selon le champ info */

 printf("Remplissage de la première liste\n");
 L1 = remplir();
 printf("Remplissage de la deuxième liste\n");
 L2 = remplir();
 printf("Affichage de la première liste\n");
 affiche(L1);

 printf("Affichage de la deuxième liste\n");
 affiche(L2);

 L3 = fusion(L1,L2);
 printf("Affichage de la liste finale");
 affiche(L3);
}
```