Listes chaînées

Description du problème :

En règle générale, pour une machine donnée, les entiers ont une longueur maximale bien spécifique, N chiffres. Par contre, parfois de grands nombres ayant des tailles supérieures à N doivent être stockés dans la mémoire. On souhaite donc pouvoir stocker et manipuler ces grands entiers positifs de longueurs indéterminées en mémoire. Pour cette raison, ces grands entiers seront représentés sous forme de chaînes de caractères dont chaque caractère représente un chiffre de ce nombre. Ensuite, ces chaînes de caractères seront représentées dans une liste chaînée en les divisant sur plusieurs maillons où chacun contient une partie de ces chaînes qui ne dépasse pas les N chiffres.

Par exemple, les nombres 4597634972106 et 587921 peuvent être représentés à raison de 5 chiffres par maillon de la façon suivante (les chiffres sont rangés par groupe de 5 à partir du début du nombre, et un maillon vide est utilisé pour séparer les nombres) :

Nous représentons ces entiers dans une liste linéaire simplement chaînée en considérant les structures suivantes :

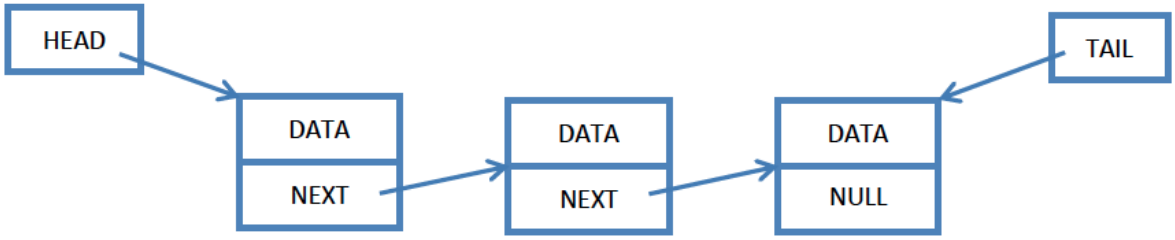
La structure Element :

- une chaîne de caractères, nommée data, de taille maximale définie par une constante N,
- un pointeur, nommé next, pointant vers l'élément suivant.

La structure List :

- un pointeur, nommé head, pointant vers le premier élément de la liste,
- un pointeur, nommé tail, pointant vers le dernier élément de la liste.

La liste sera de la forme suivante :



AI01 : TP Listes chaînées

Fonctions à implémenter :

1- **void initialize (List *list)** : pour créer et initialiser une liste vide.

2- **void insert_empty_list (List *list, char *str)** : pour insérer un nombre représenté par la chaîne de caractères *str* dans la liste chaînée *list* si la liste est initialement vide.

3- **void insert_begining_list (List *list, char *str)** : pour insérer un nombre représenté par la chaîne de caractères *str* au début de la liste chaînée *list* si la liste contient initialement des éléments.

4- **void insert_end_list (List *list, char *str)** : pour insérer un nombre représenté par la chaîne de caractères *str* à la fin de la liste *list* si la liste contient initialement des éléments.

5- **int insert_after_position (List *list, char *str, int p)** : pour insérer un nombre représenté par la chaîne de caractères *str* après le *p*ème nombre dans la liste si la liste contient initialement des éléments. Cette fonction retourne 0 si l'insertion a été réalisée et -1 en cas d'échec (si la position du nombre à ajouter, indiqué par l'utilisateur, n'existe pas dans la liste).

6- **int remove (List *list, int p)** : pour supprimer le *p*ème nombre représenté par une chaîne de caractères dans la liste si la liste contient initialement des éléments. Cette fonction retourne 0 si la suppression a été réalisée et -1 en cas d'échec (si la liste est vide ou si la position du nombre à supprimer, indiqué par l'utilisateur, n'existe pas dans la liste).

7- **int compare (char *str1, char *str2)** : pour comparer les deux nombres représentés par les chaînes de caractères *str1* et *str2*. Cette fonction retourne 1 si le premier nombre représenté par *str1* est plus grand que le deuxième représenté par *str2*, et retourne 2 si c'est l'autre cas.

8- **int sort (List *list)** : pour trier par ordre croissant les nombres représentés sous forme de chaîne de caractères dans la liste. Cette fonction retourne 0 si le tri a été réalisé et -1 en cas d'échec (si la liste est vide).

9- **void display (List *list)** : pour afficher les nombres représentés dans la liste. Cette fonction doit afficher « EMPTY LIST » si la liste est vide.

10- **void destruct (List *list)** : qui détruit la liste toute entière.

Programme principal :

Utiliser les fonctions développées pour écrire le programme *main* qui permet à l'utilisateur de remplir et de manipuler sa propre liste en lui offrant les possibilités suivantes :

1- ajouter un nombre en début de liste

2- ajouter un nombre en fin de liste

3- ajouter un nombre à une certaine position dans la liste

4- supprimer un nombre d'une certaine position de la liste

5- trier la liste par ordre croissant

6- afficher la liste

7- détruire la liste toute entière

8- quitter

Autres consignes :

A chaque itération on doit :

1- afficher la nouvelle liste obtenue

2- donner toutes les possibilités à l'utilisateur pour choisir la nouvelle opération

3- récupérer le choix de l'utilisateur

4- Exécuter la demande de l'utilisateur si c'est possible. Si ce n'est pas possible de l'exécuter, afficher la cause.

Bonus :

Ajouter la fonction **void sum (List *list)** qui calcule la somme des nombres représentés dans la liste et ajoute le résultat à la fin de la liste.

Dossiers à déposer :

L'organisation **MINIMALE** du projet sous Visual Studio ou CodeBlocks est la suivante :

- Fichier d'en-tête tp3.h, contenant la déclaration des structures/fonctions de base,
- Fichier source tp3.c, contenant la définition de chaque fonction,
- Fichier source main.c, contenant le programme principal.

Avec ces trois fichiers, un compte rendu doit être présenté, comportant principalement :

- les spécifications précises de chaque fonction : ses entrées, ses sorties (pas de code dans le compte rendu),
- les complexités des différentes procédures réalisées,
- une réflexion sur le projet avec des idées d'amélioration.

Le compte rendu ne doit pas dépasser les 3 pages.