Éditeur de texte simplifié (opérations avancées sur chaînes)

Réalisé par :

-Aimrane Alyachiri Appogee :2334282
- Amine Ait Saleh Appogee : 2228289
- Oussama Chekroun Appogee :2229322

1. Problématique du projet

Dans le cadre de notre module, il nous a été demandé de développer un éditeur de texte simplifié permettant de :

- Gérer dynamiquement un texte ligne par ligne.
- Insérer et supprimer des lignes à des positions précises.
- Rechercher et remplacer des mots dans tout le texte.
- Revenir en arrière sur les dernières modifications (fonction Undo).

L'objectif principal est de manipuler efficacement les chaînes de caractères dynamiques et de mettre en œuvre des structures de données telles que les listes chaînées, les piles et les files.

2. Fonctionnement du code

2.1Stockage du texte

Le texte est représenté sous forme de file, où chaque élément de la file correspond à une ligne. Cette file est implémentée à l'aide d'une **liste chaînée** dynamique, permettant une gestion flexible du nombre de lignes sans limitation fixe. Chaque nœud contient une chaîne de caractères (ligne de texte) et un pointeur vers la ligne suivante.

2.2 Ajout d'une ligne

La fonction **AjouterLigne()** permet d'ajouter une ligne dans un texte en spécifiant la ligne à insérer et la position dans le texte, si la position ne pas dans le texte affiche un message «la position invalide», Elle Crée une nœud , Parcourir le Texte pour insérer ce nœud a la position indiquée , puis copier le texte fourni dans le nœud a l'aide de la fonction **strcpy()** de la bibliothèque **<string.h>,** puis relié les Nœud a cote s'ils existent pour maintenir le continuité de la structure de la file**.**

2.3 Suppression

La fonction **SupprimeLigne()** est utilisé pour supprime une ligne dans un texte, en spécifiant la position de la ligne dans le texte si la position n'existe pas dans le texte affiche message « la position invalide », La fonction parcourt le File pour trouve et supprimer dans la position indiquée, ensuite relie les Nœud adjacent, si 'ils existent afin de maintenir le contuité de la structure de la file.

2.4 Recherche

Une fonction parcourant l'ensemble des lignes d'une file/liste chainée et Recherche toutes les occurences d'un mot donné. Elle affiche chaque ligne avec les occurences du mot en jaune. Si aucune occurence n'est trouvée dans toute la file, elle affiche un message d'erreur en rouge

2.5 Remplacement

La fonction remplace() est une fonction qui remplace toutes les occurrences d'un mot (**motAncian**) par un nouveau mot (**motnouveau**) via parcours de chaque ligne de la file et comptage des occurrences de le ancien mot , Si elle repère des occurrences, cette fonction affecte dynamiquement une nouvelle chaîne à la taille adéquate en fonction de la différence de longueur entre le contenu des deux chaînes. Puis, elle recopie le texte en remplaçant tous les mots égaux à (**ancienMot**) par le nouveau mot qu'est le mot (**nouveauMot**), elle libère l'ancien texte et remplace l'ancienne chaîne par la nouvelle copie où ont été fait les remplacements. Enfin, la fonction retourne 1 si un remplacement a bien eu lieu, sinon elle retourne 0.

2.6 Undo

La **pile (Pile)** enregistre toutes les modifications apportées au texte : type d'action, position, ancienne valeur, nouvelle valeur. Lorsque la fonction **undo()** est appelée, elle récupère la dernière action effectuée et appelle automatiquement la fonction inverse correspondante *(par exemple : annuler un ajout = faire une suppression, annuler une suppression = refaire l'ajout, annuler un remplacement = rétablir l'ancien mot)*.

Cela permet de restaurer l'état précédent du texte et d'expérimenter librement sans risque de perdre des données.

7.2 Fonction principale main()

La fonction **main()** représente le cœur de l'application. Elle est responsable de l'initialisation des structures de données principales : une file pour stocker dynamiquement le texte ligne par ligne, et une pile pour gérer les opérations d'annulation (Undo).

Une fois l'environnement prêt, **main()** entre dans une boucle principale interactive qui écoute les entrées clavier à l'aide de **getch()**. Chaque touche saisie déclenche une action spécifique grâce à un switch-case, en fonction du raccourci utilisé :

- **^A** → Ajouter une ligne à une position choisie.
- **^D** → Supprimer une ligne existante.
- **^S** → Rechercher un mot dans le texte et en afficher les occurrences.
- **^R** → Remplacer un mot par un autre dans tout le texte.
- **^Z** → Undo, pour annuler la dernière modification (ajout, suppression ou remplacement).
- **^X** → Quitter le programme en libérant la mémoire utilisée.

Chaque opération effectuée est suivie d'un rafraîchissement de l'affichage (**system("cls")**) et parfois d'une pause (**Sleep()**) pour rendre l'interface plus fluide et réactive.

Généralement , la fonction **main()** joue le rôle d'interface utilisateur tout en assurant le contrôle logique de l'éditeur de texte.

3. Difficultés rencontrées

- Gestion de la mémoire dynamique (`malloc`, `strdup`, `free`) pour éviter les fuites.

- Implémentation correcte de l'Undo en stockant les informations nécessaires selon le type de modification.
- Recherche de tous les occurrences d'un mot .
- Modifier tous les occurrences d'un mot .

4. Conclusion

Ce projet nous a permis de mettre en pratique la manipulation avancée de chaînes de caractères en C ainsi que les structures de données fondamentales. L'approche modulaire et la gestion des raccourcis clavier offrent une expérience utilisateur fluide et intuitive.