

Rapport du projet de l’Algorithmique

Gérer un texte.



Etudiant: Ilyes Yamoun.

Novembre/Decembre 2017

Sommaire:

1. Introduction.
2. Plan du projet.
3. Fonctions utilisées ( déja existantes ).
4. Fonctions crées:

4) 1. myHeader.h

4) 2. listeLin.h

4) 3. listeLinBi.h

4) 4. listeLinBiC.h

4) 5. pile.h

4) 6. file.h

1. Le main.
2. Conclusion.
3. References.
4. Introduction:

Ce projet Algorithmique porte sur l’idée de créer un éditeur de texte qui est un logiciel qui permet à l’utilisateur d’introduire un texte et le modifier ou faire des vérifications sur ce dernier grâce au menu qui lui est offert, il pourra ensuite l’enregistrer s’il le désire.

1. Plan du projet:

J’ai devisé ce projet en plusieurs fichiers .c dont les entêtes sont enregistrés dans les fichiers .h

Les fichiers .c :

myHeader.c :

Contient les fonctions demandées dans la patie 1 du projet.

listeLin.c :

Contient les fonctions liées à la structure de données : Liste chainée.

listeLinBi.c :

Contient les fonctions liées à la structure de données : Liste chainée bidirectionnelle.

listeLinBiC.c :

Contient les fonctions liées à la structure de données : Liste chainée bidirectionnelle circulaire.

pile.c :

Contient les fonctions liées à la structure de données : Pile (Stack).

file.c :

Contient les fonctions liées à la structure de données : File (Queue).

1. Fonctions utilisées (déja existantes):

J’ai utilisé des fonctions disponibles sur les bibliothèques du C, je cite :

malloc() et free().

Pour gérer les pointeurs.

Et :

memset(),strcpy(),strncpy() et strcmp().

Pour gérer les chaines de caractères.

Ainsi que system() pour exécuter des commandes dans la console, exemple : system(« cls ») pour effacer le contenu actuel de la console.

1. Fonctions crées:

4 )1. myHeader.h :

myHeader.h contient les entetes des fonctions demandées dans la partie 1 du projet :

texte creer();

void liberer(texte \*t);

int taille(texte t);

int capacite(texte t);

char caractere(texte t, int pos);

void modifierCapacite(texte \*t, int minimum);

void modifierCapacite2(texte \*t, int minimum);

void ajouterCaractere(texte \*t, char caractere);

void ajouterChaine(texte \*t, char \*str);

int position(texte t, char \*str);

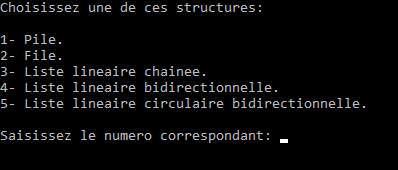
int dernierePosition(texte t, char \*str);

void insererChaine(texte \*t, char \*str, int debut);

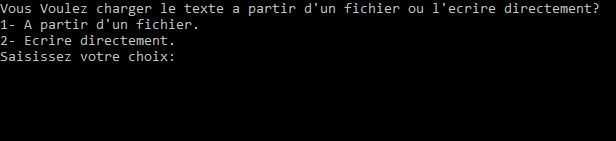
Ainsi que des fonctions reliées aux menus que j’ai créé pour traiter les actions de l’utilisateur :

void cls() : sert à supprimer le contenu actuel de la console.

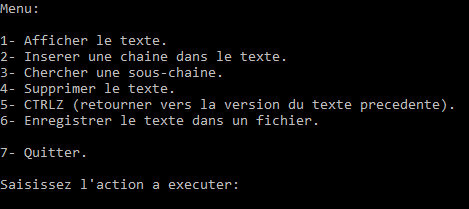
int chooseStruct() : affiche un menu pour choisir le type de la structure. Et le retourne en sortie.



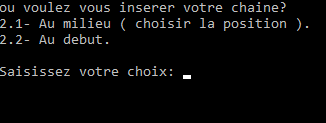
int chooseMethod() : affiche un menu pour choisir le type du chargement du texte ( depuis un fichier ou bien direct). Et le retourne en sortie.



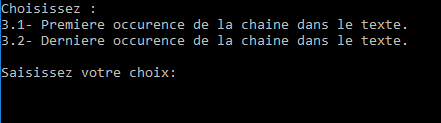
int chooseMenu() : affiche le menu principale. Et le retourne en sortie.



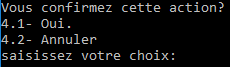
int chooseSubMenu2() : affiche un sous-menu lors de l’insertion d’une chaine de caractère ou d’un seul caractère dans le texte. Et le retourne en sortie.



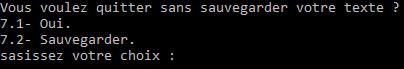
int chooseSubMenu3() : affiche un sous-menu lors de la recherche d’une chaine de caractère ou d’un seul caractère dans le texte. Et le retourne en sortie.



int chooseSubMenu4() : affiche un sous-menu lors de la suppression du texte, offrant à l’utilisateur le choix entre confirmer ou annuler son action. Et le retourne en sortie.



int chooseSubMenu7() : affiche un sous-menu lorsque l’utilisateur veut quitter l’application et qu’il n’a pas enregistré son texte après quelconque modification sur ce dernier. Et le retourne en sortie.



4 )2. listeLin.h :

Contient les fonctions liées à la structure de données : Liste chainée.

listeLin\* creerListeLin() : créer un élément de la liste chainée.

int tailleListeLin(listeLin \*tete) : retourne la taille de la liste chainée.

int listeLinVide(listeLin \*tete) : retourne 1 si la liste chainée est vide, 0 sinon.

int listeLinPleine(listeLin \*tete) : retourne 1 si la taille de la liste chainée est égale à 10, 0 sinon.

listeLin\* supprimerFinListeLin(listeLin \*tete) : supprime le dernier élément de la liste chainée. Et retourne la tete en sortie (pour exécuter les modifications).

listeLin\* supprimerDebutListeLin(listeLin \*tete) : supprime le premier élément de la liste chainée. Et retourne la tete en sortie (pour exécuter les modifications).

listeLin\* ajouterFinListeLin(listeLin \*tete, texte \*elem) : ajouter un élément à la fin de la liste chainée contenant comme texte celui qui est dans « elem ». Et retourne la tete en sortie (pour exécuter les modifications).

listeLin\* ajouterDebutListeLin(listeLin \*tete, texte \*elem) : ajouter un élément au début de la liste chainée contenant comme texte celui qui est dans « elem ». Et retourne la tete en sortie (pour exécuter les modifications).

texte\* retournerPosListeLin(listeLin \*tete, int pos) : retourne le texte à la postion « pos » donnée.

void afficherListeLin(listeLin \*tete) : afficher tous les textes présents dans la liste chainée.

listeLin\* ctrlzListeLin(listeLin \*tete) : revient une version en arrière du texte dans la liste chainée ( supprime le dernier élément ).

4 )3. listeLinBi.h :

listeLinBi\* creerListeLinBi() : créer un élément de la liste chainée bidirectionnelle.

int tailleListeLinBi(listeLinBi \*tete) : retourne la taille de la liste chainée bidirectionnelle.

int listeLinBiVide(listeLinBi \*tete) : retourne 1 si la liste est vide, 0 sinon.

int listeLinBiPleine(listeLinBi \*tete) : retourne 1 si la taille de la liste est égale à 10, 0 sinon.

listeLinBi\* supprimerFinListeLinBi(listeLinBi \*tete) : supprime l’élément qui se situe à la fin de la liste.

listeLinBi\* supprimerDebutListeLinBi(listeLinBi \*tete) : supprime l’élément qui se situe au début de la liste.

listeLinBi\* ajouterFinListeLinBi(listeLinBi \*tete, texte \*elem) : ajoute un élément à la fin de la liste contenant de le texte contenu dans « elem ».

listeLinBi\* ajouterDebutListeLinBi(listeLinBi \*tete, texte \*elem) : ajoute un élément au début de la liste contenant de le texte contenu dans « elem ».

texte\* retournerTexteListeLinBi(listeLinBi \*tete, int pos) : retourne le texte qui se situe dans la position « pos » donnée.

void afficherListeLinBi(listeLinBi \*tete) : affiche tous les textes qui existent dans la liste.

listeLinBi\* ctrlzListeLinBi(listeLinBi \*tete) : revient une version en arrière du texte dans la liste (supprime le dernier élément).

4 )4. listeLinBiC.h :

int tailleLisetLinBiC(listeLinBi \*tete) : retoune la taille de la liste.

int listeLinBiCPleine(listeLinBi \*tete) : retourne 1 si la taille de la liste est égale à 10, 0 sinon.

listeLinBi\* supprimerDebutListeLinBiC(listeLinBi \*tete) : supprime l’élément qui se situe dans le début de la liste.

listeLinBi\* supprimerFinListeLinBiC(listeLinBi \*tete) : supprime l’élément qui se situe dans la fin de la liste.

listeLinBi\* ajouterDebutListeLinBiC(listeLinBi \*tete, texte \*elem) : ajouter une élément au début de la liste qui contient comme texte celui qui est dans « elem ».

listeLinBi\* ajouterFinListeLinBiC(listeLinBi \*tete, texte \*elem) : ajouter une élément en fin de liste qui contient le texte qui est contenu dans « elem ».

texte\* retournerTexteListeLinBiC(listeLinBi \*tete, int pos) : retourne le texte qui se situe à la position « pos » dans la liste.

void afficherListeLinBiC(listeLinBi \*tete) : affiche tous les élément contenus dans la liste.

listeLinBi\* ctrlzListeLinBiC(listeLinBi \*tete) revient une version en arrière du texte dans la liste (supprime le dernier élément).

4 )5. Pile.h :

pile initPile() : initialiser la pile.

int taillePile(pile \*maPile) : retourne la taille de la pile.

int pileVide(pile \*maPile) : retoune 1 si la pile est vide, 0 sinon.

int pilePleine(pile \*maPile) retourne 1 si la taille de la pile est égale à 10, 0 sinon.

texte SommetPile(pile \*maPile) : retourne le texte se trouvant au sommet de la pile.

texte depiler(pile \*maPile) : retourne le texte se trouvant au sommet de la pile et le supprime de la pile.

void empiler(pile \*maPile, texte \*elem) : ajoute un texte en haut du sommet actuel contenant un « elem » comme texte.

void afficherPile(pile maPile) : affiche tous les élément contenus dans la Pile.

void ctrlzPile(pile \*maPile) : revient une version en arrière du texte dans la pile (dépile l’élément qui se trouve au sommet de la Pile).

4 )6. File.h :

file initFile() : initialise la file.

int fileVide(file \*maFile) : retourne 1 si la file est vide, 0 sinon.

texte \*teteFile(file \*maFile) : retourne le texte qui se trouve en tete de la file.

int filePleine(file \*maFile) : retoune 1 si la taille de la file est égale à 10, 0 sinon.

texte \*defile(file \*maFile) : retoune le texte qui se trouve en tete de file, et le supprime ensuite.

void enfiler(file \*maFile, texte \*elem) : ajouter un élément vers la fin de la file, contenant le contenu de « elem ».

void afficherFile(file maFile) : affiche tous les élément contenus dans la File.

void ctrlzFile(file \*maFile) : revient une version en arrière du texte dans la file.

1. Le main:

La première partie du « main » n’est qu’une déclaration des variables et des structures dont on a besoin dans la mémoire, ainsi qu’un pointeur sur un fichier.

1. On donne la main à l’utilisateur pour choisir la structure de donnée qui lui convient : choiceStruct = chooseStruct();
2. One donne la main à l’utilisateur pour choisir la méthode avec laquelle il veut écrire son texte (1-fichier, 2- directement) : choiceLoad = chooseMethod();
3. Reserver de l’espace mémoire selon la structure voulue : switch(choiceStruct).. ligne 72

4-Afficher le menu principal et donne la main à l’utilisateur pour choisir l’action qu’il veut réaliser sur son texte. Après avoir exécuté l’action voulue, on affiche « Tache exécutée. » si l’action a été exécuter avec succès, « Erreur! » sinon.

L’exécution des taches est faite dans une boucle do{}while() , grâce à des switch(){cases} ;

1. Conclusion:

Dans ce projet j’ai implémenté les structures que j’ai cité dans ce rapport en utilisant le principe « LIFO » pour optimiser le temps d’exécution et de recherche de la dernière version du texte dans la structure choisie par l’utilisateur, et aussi pour optimiser l’utilisation de CTRLZ.  
J’ai aussi utilisé une fonction CTRLZ pour chacune des instructions séparément.

Çe projet m’a permis d’apprendre davantage sur la bibliothèque <string.h> et l’utilisation des pointeurs.

1. References:

Le site Stack overflow pour résoudre les bugs que j’ai rencontré lors de la programmation du projet : <https://stackoverflow.com/> .

Le site Tutorials Point pour apprendre de nouvelles fonctions et leurs utilisations : <https://www.tutorialspoint.com/> .

Fin.