ETAPE 1 : CORRECTEUR ORTHOGRAPHIQUE

RAPPORT

SOMMAIRE :

INTRODUCTION :

Lors de cette première étape, nous verrons comment implémenter un premier correcteur orthographique qui recherche des mots mal orthographiés. En effet, ce premier correcteur prend un texte à corriger et un dictionnaire en paramètres. Puis il renvoie les mots qui n'appartiennent pas à ce dictionnaire. Ainsi, nous verrons comment implémenter un arbre ternaire de recherche et une liste qui contient tous les mots mal orthographiés. Nous manipulerons donc les listes chainées, les types structurés et la modularisation de divers fichiers ainsi que la compilation avec un Makefile.

1) LES LISTES

Premièrement, nous avons découpés le projet en diverses modules. En effet, il y a 3 modules différents. Un module Listes dans lequel on retrouve toutes les fonctions pour gérer les listes, un module ATR dans lequel on retrouve toutes les fonctions qui concernent les arbres ternaires de recherche. Puis le module Main, dans lequel on retrouve la fonction main qui gère le projet.

Deuxièmement, pour la gestion des listes on a tous d'abord définie une structure liste qui prend comment paramètres un mot et la structure vers la prochaine cellule. Puis on a écrit une première fonction pour allouer une cellule dans la liste chainée. Pour cela, on a alloué un espace mémoire avec la fonction malloc puis ajouter le mot à cette espace. Ensuite, on a écrit

une fonction pour insérer un mot dans une liste. Pour cela, il suffit d'allouer une cellule avec la fonction précédente et d'ajouter cette cellule en tête de liste. C'est-à dire de faire pointer le champ « next » de la cellule vers la liste.

D'autre part, on a écrit une fonction pour libérer la liste, c'est-à-dire de libérer la mémoire qui contient cette liste. On a aussi écrit une fonction qui affiche la liste en parcourant cette cellule de la liste et en affichant le mot. Enfin, on a écrit une fonction pour savoir si un mot appartient à une liste ou non, en prenant un mot et en regardant si le mot est égal à un des mots présent dans cette liste.

2) LES ARBRES TERNAIRES DE RECHERCHE

Dans cette partie, nous verrons comment nous avons implémenter les arbres ternaires de recherche. En effet, un plus des fonctions principales demandée nous avons dédoublés certaines fonctions avec des fonctions auxiliaire et nous avons écrit de nouvelles fonctions comment l'allocation de cellule dans l'arbre ou savoir si un mot est dans l'arbre et l'ajout d'une branche entière.

Premièrement, on a écrit une fonction pour créer un ATR vide. Il suffit de créer un arbre vide et de renvoyer cette arbre vide.

Deuxièmement, on a dû ajouter des mots dans l'arbre. Pour cela, on a donc dû allouer des cellules pour chaque lettre de l'arbre. Il suffit de procéder de la même manière que pour les listes mais dans le cas de l'ATR.

Troisièmement, on a écrit deux fonctions pour libérer l'arbre. En effet, on a dû écrire une fonction auxiliaire qui libère juste une partie de l'arbre par exemple une branche et ses sous-branche et une fonction qui libère l'arbre entier.

Quatrièmement, on écrit une fonction pour savoir si un mot était déjà dans l'arbre. Cette fonction ce base sur le principe de recherche d'un mot dans l'arbre.

Cinquièmement, on a dû écrire deux fonctions pour l'insertion de mot dans l'arbre. En effet, nous devions prendre en compte le moment ou un mot n'existant pas ou que l'arbre était vide. Dans ce cas, on ajoutait une branche entière en fils milieu. Sinon, si une partie du mot existe déjà ou alors que l'arbre n'est pas vide. Il faut parcourir l'arbre en regardant si la lettre d'ajout existe ou pas ou sinon en comparant suivant l'ordre alphabétique.

Sixièmement, on a deux fonctions pour la suppression. En effet, pour cette fonction nous avions besoins de renvoyer une valeur soit 1, 2 ou 0. Pour cela, nous avions opter pour une fonction auxiliaire.

Dans cette fonction on parcourt l'arbre de la même manière que l'insertion même si on arrive sur la fin du mot que l'on veut supprimer alors on regarde si le il n'a pas de fil droit si c'est le

cas alors on supprime juste la cellule et on remplace la cellule par son fils droit. Sinon on remonte à son père. Et on supprime celui-ci.

Enfin, on a écrit une fonction qui permet d'afficher l'arbre. On utilise un tableau de caractères dans le quelle on place les lettres du mot lors que parcours de l'arbre. Puis on affiche le mot.

3) LE MAIN

Dans cette partie, nous verrons comment nous avons implémenter la fonction principale du projet.

Tous d'abord, on a géré l'ouverture des fichiers passés en paramètres ainsi que leurs lectures. Puis, pour chaque mot présent dans le dictionnaire, on insère celui-ci dans l'arbre. Enfin, pour chaque mot présent dans le texte à corriger, on regarde si le mot est dans l'arbre si ce n'est pas le cas, alors on insère le mot dans la liste seulement s'il n'est pas présent dans la liste.

CONCLUSION :

Lors de ce projet, nous avons pu mettre en place une première version d'un correcteur orthographique. Ainsi, nous avons tous d'abord modulariser le projet, puis nous avons mis en place des listes chainées simple. Et pour finir, nous avons mis en place des arbres binaires de recherche. Notre projet est fonctionnel et réponds aux demandent de l'énoncé. Nous avons conscience que certaines fonctions peuvent-être amélioré.

ANNEXE 1 : (Répartition du travail)

Pour l'organisation du travail, la communication s'est passée en majorité via Discord. Avant d'écrire les fonctions, nous avons pré-alloué quelle fonction l'un des deux de nôtre binôme va écrire.

Le Makefile a été écris par Gaël, et une grande partie du main a été pris en charge pas Hervé. Les modules ont une parité de contribution assez uniforme, certaines fonctions ont été pris en charge pas l'un du binôme et les autres fonctions ont été écris par l'autre.

Il n'y a pas eu de chevauchement ou de gros désaccord en termes de compréhension au niveau du code entre nous deux, avant d'écrire les fonctions assez sensibles il y eu des échanges ce qui permet de ne pas rencontrer des difficultés vers l'écriture du main.

Il y a eu quelques erreurs de segmentations après que la première version du main, mais ils ont été résolus assez rapidement grâce à l'outil de debug llbd / gbd.

ANNEXE 2 : (Compilation du programme)

Pour compiler le projet, nous avons utilisé un Makefile qui permet de compiler les divers modules. Pour cela, il faut utiliser les commandes suivantes :

- make correcteur_0 (pour compiler le projet correcteur_0)
- ./correcteur_0 a_corriger_0.txt dico_1.dico (pour exécuter le projet correcteur_0)

ANNEXE 3 : (Utilisation des fichiers)

Pour utiliser le projet rien n'est demandé à part compiler et exécuter le projet. Pour cela, veuillez vous rendre à l'annexe 1 qui explique comment compiler et exécuter le programme. Pour l'exécution, il est possible de choisir entre deux textes (a_corriger_0.txt, a_corriger_1.txt) et 3 dictionnaires (dico_1.dico, dico_2.dico, dico_3.dico).