JEAN-LOUIS
Baptiste
IMA3

Rapport du projet de programmation avancée en langage C

année scolaire : 2018-2019

Page

Sommaire

Structuration des données

Un dictionnaire (dico) est un tableau de 26 arbres. Il y a 26 pointeurs car il y a 26 lettres dans l’alphabet utilisé par l’anglais.

Un arbre (tree) est un pointeur de nœud.

Un nœud (node) est composé d’un char, qui est la lettre qu’il doit stocker, d’un byte, définissant s’il existe un mot s’arrêtent à lui, et de quelle forme il sera, et d’un tableau de 26 arbres contenant les lettres de la potentielle suite du mot.

Structuration du programme

I) Dans treeh.c

make_empty_tree(), make_empty_node(), make_node(), delete_tree, is_empty(), is_followed() sont les fonctions classiques pour un arbre, is_followed() étant la négation de la plus classique !is_leaf(). Les fonctions de la parties endKind sont utiles pour l'affichage du dictionnaire (voir la fonction print dans dico.c) . Dans la partie other, on trouve les fonctions hash(), ischar_of_word(), is_word() et end_kind(). Hash() calcule à quel position doit être mis le pointeur vers le nœud suivant en fonction de la lettre qu'il stockera. ischar_of_word() dis si le caractère passé en paramètre peut être présent dans un mot du dictionnaire. end_Kind() calcule la valeur du byte endKind qui sera présent dans le nœud le plus loin de la racine, is_word() dit si la valeur de ce byte correspond à une fin de mot ou non. Autrement dit c'est dans end_kind() qu'est présente la définition informatique du mot, que j'ai utilisé.

II) Dans dico.c

On trouve les fonctions d'initialisation et de destruction du dictionnaire, une fonction d'ajout de mot (addto_dico()), et une qui nous dis si un mot est présent dans le dictionnaire. loadfrom_file() permet de rajouter tous les mots du fichiers dans le dictionnaire. printto_file() permet d'afficher dans le fichier passé en paramètre tous les mots du dictionnaire, pour ce faire elle appelle la fonction récursive print(). Enfin test_words() liste explicitement, dans le fichier en sortie, chacun des mots présents dans le fichier en entrée et absents du dictionnaire. Pour chacun des mots listé, s'il correspond à la définition d'un mot, il est enregistré dans une liste de mots, qui une fois la lecture du fichier en entrée terminé, propose à l'utilisateur de les ajouter au dictionnaire ou non .

III) Dans functions.c

On trouve des fonctions d'usage général, telle que clear() permettant d'effacer l'écran de terminal, pause() qui attend que l'utilisateur tape <return>, saisie() qui permet de saisir un entier dans un intervalle défini par min et max, et ctoi() qui permet de transformer chaque caractère numérique en sa valeur entière ('0' donne 0).
On trouve également une structure LIFO (implantée par une liste chaînée) de chaînes de caractères, les typedefs et les fonctions associées à son utilisation (voir test_words() dans dico.c).

IV) Dans main.c

On trouve getfile() et le main(). getfile() est une fonction de saisie qui retourne un FILE* vers ce qui deviendra le fichier d'entrée ou de sortie suivant le mode passé en paramètre. Dans le main() on trouve une initialisation et une boucle contenant le menu.

Cahier des charges et limites de la solution

I) Définition d’un mot

Un mot dans sa totalité ne pourra pas dépasser 29 caractères.
La définition que j’utilise est « * » ou « *’s », avec « * » un ensemble de caractère alphabétiques tel que les lettres soient : toutes minuscules (répond vrai à is_common_end()), toutes majuscules(répond vrai à is_acronyme_end()), ou la première est majuscule et les suivantes minuscules (répond vrai à is_proper_end()).

II) Ensemble de caractères

J’utilise une simple table ASCII 7bits ce qui signifie que mon dictionnaire n’accepte pas les caractère avec accent ou cédille. Si vous rentrez un de ces caractères, le comportement de mon programme est indéfini. Il semblerait néanmoins qu’il survive au dictionnaire présent dans le fichier « ./img/amrican-english ».