Projet de Programmation Avancée
Correcteur orthographique

L'objectif de notre projet était de réaliser un programme permettant de détecter dans un texte tous les mots mal orthographiés. Pour cela, on utilisera un dictionnaire qui sera construit à partir d'un texte ou d'un ensemble de mots.

Afin de minimiser l'espace mémoire nécessaire au stockage du dictionnaire tout en fournissant un temps de recherche bas, la structure de données que nous avons utilisée est un arbre préfix. Il s'agit d'une structure arborescente pour laquelle des mots ayant des préfixes communs sont factorisés : chaque nœud est une lettre qui peut être la dernière d'un mot, ou pas.

Notre projet devait respecter le cahier des charges suivant :

- Définir et implémenter une structure de données permettant de stocker et de manipuler un dictionnaire sous forme d'arbre préfix ;
- Charger un dictionnaire à partir d'un fichier texte de données ;
- Analyser l'orthographe d'une phrase ou d'un texte, et indiquer le nombre de mots qui ne sont pas présents dans le dictionnaire.

Nous avons choisi d'ajouter un correcteur orthographique qui sera expliqué dans les détails ci-dessous.

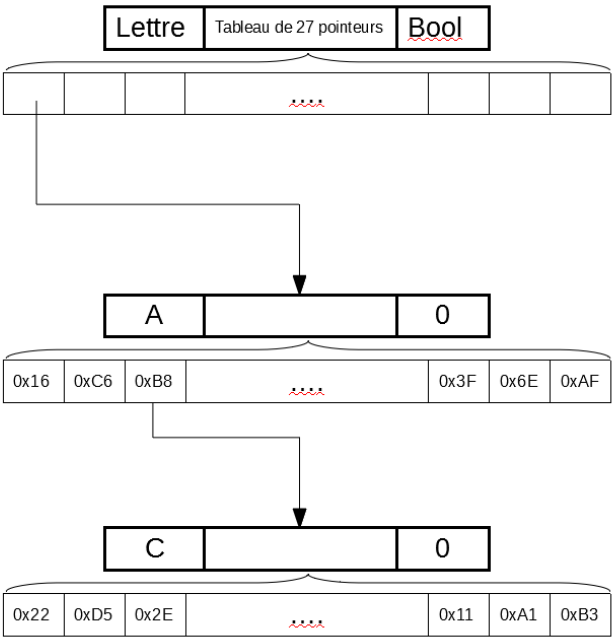
Nous allons suivre ces différentes rubriques :

I. La structure de données

Nous avons défini un nouveau type de données nommé 'cell' représentant une cellule de données. Ce type est en réalité une structure comportant plusieurs éléments :

- Un char contenant la lettre stockée ;
- Un tableau de pointeurs de 'cell' (27 cases : on considère l'alphabet et l'apostrophe) ;
- Un booléen indiquant si la lettre est terminale.

Voici un schéma explicatif de notre structure.



Nous avons choisi ce type de structure pour plusieurs raisons :

- La facilité de compréhension et la lisibilité ;
- Le lien facile entre un caractère et sa place dans le tableau de pointeurs ;
- Il suffit de chercher si la case censée contenir la cellule d'un caractère pointe sur NULL pour vérifier l'existence de celle-ci. Avec une liste chaînée, on aurait dû parcourir la liste dans sa totalité pour le même résultat.

II. Fonctionnement général

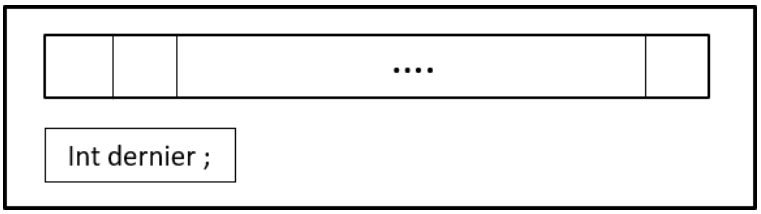
Avant de vous expliquer en détails le fonctionnement des algorithmes principaux, nous allons décrire le fonctionnement général du programme.

Dans un premier temps, après avoir procédé à la récupération dans argv[] et l'ouverture des fichiers (bibliothèque et fichier à analyser), nous devons lire le fichier de bibliothèque et implémenter notre arbre préfix.

Une fois tous les mots lus et correctement entrés dans la structure, nous passons à l'analyse du second fichier. Celui-ci est lu mot à mot de la même manière que la bibliothèque. Les mots sont ensuite analysés pour savoir s'ils sont présents ou non dans l'arbre et donc dans le dictionnaire.

Si une erreur est détectée, une fonction permet d'incrémenter un compteur d'erreurs et lance le processus de correction.

Cette fonction va créer une liste contigüe qui permettra le stockage des mots possibles pour la correction. Une autre fonction va remplir cette liste avec les différents mots, en se basant sur la partie du mot qui est correcte.



Liste contigüe

L'utilisateur devra alors choisir s'il veut corriger le mot, et si c'est le cas, quel mot il veut comme correction. Ce mot sera envoyé à une dernière fonction qui effectue le remplacement.

Pour remplacer un mot comportant une erreur par un autre, nous avons utilisé un fichier tampon qui stocke le début du texte jusqu'au mot avec erreur, puis le mot de correction, et enfin la fin du texte sans le mot avec erreur. Ce fichier tampon est recopié dans le fichier original et est supprimé.

Une fois ces étapes réalisées, on lance le scan d'un nouveau mot. A la fin du programme, on affiche le nombre de fautes dans le fichier et le nombre de corrections. Puis, on libère l'espace mémoire utilisé par l'arbre préfix et on ferme les fichiers.

Maintenant que vous avez compris le fonctionnement général de notre programme, voici l'explication des algorithmes principaux.

III. Explication des algorithmes principaux

Nous allons vous expliquer les algorithmes des fonctions principales dans l'ordre de leur utilisation. Commençons donc avec la lecture des différents fichiers, qui suivent le même algorithme.

Ces fonctions lisent mot par mot les fichiers à l'aide de la fonction 'scanf' et les envoient aux fonctions d'ajout et de scan s'ils ne comportent pas d'accent.

Au cours de la lecture du fichier bibliothèque, il faut implémenter l'arbre préfix. Pour cela, plusieurs étapes sont effectuées :

- On commence par initialiser l'arbre en créant une première cellule qui servira de racine pour tous les mots. Elle ne contiendra pas de lettre mais le caractère de code Ascii 7 qui n'aura aucune utilité.
- On va ensuite lire les mots un par un dans le dictionnaire et vérifier qu'ils ne comportent pas d'accent (explications plus bas). Chaque mot, une fois lu, et envoyé dans une fonction d'ajout.

- La fonction d'ajout va ajouter le mot lettre par lettre, en considérant 2 cas : l'existence de la cellule car un mot a déjà cette lettre comme racine, et l'absence de cellule.
  Dans le premier cas, on se déplace dans cette nouvelle cellule afin d'ajouter la lettre suivante.
  Dans le second cas, on crée une nouvelle cellule et on fait pointer la précédente vers celle-ci, dans laquelle on se déplace également.
- Sans modifier l'état 'vrai' d'une lettre terminale existante, on modifie le booléen de fin de mot à 'faux' sauf si on est arrivé à la fin du mot.

Une fois tous les mots lus et correctement entrés dans la structure, nous passons à l'analyse du second fichier, en plusieurs étapes également :

- Au cours de la lecture, chaque mot est envoyé à la fonction 'scan_word'. Cette fonction va analyser les mots est confirmer (ou non) qu'ils font bien partie du dictionnaire.
- La fonction va parcourir le mot à analyser (tant qu'on n'arrive pas à '\0'). Chaque caractère va recevoir son indice de rangement dans le tableau de pointeurs présent dans les cellules de l'arbre. La fonction va donc tester si le pointeur vers la cellule censée contenir la lettre est NULL ou non.
  S'il n'est pas NULL, on se place dans cette cellule et on passe à la lettre suivante.
  S'il est NULL, on lance la fonction d'ajout d'erreur car la lettre n'a pas été trouvée.
- Si on arrive à la fin du mot à analyser mais pas à la fin du mot dans l'arbre, c'est que le mot à analyser est trop court pour être bon. On lance donc également la fonction d'ajout d'erreur.

Lorsqu'une erreur est détectée, la fonction 'add_error' est lancée. Cette fonction permet 2 choses :

- Elle imprime, sur la sortie standard, l'erreur trouvée et incrémente le nombre d'erreurs.
- Elle lance le processus de correction si l'erreur n'est pas la première lettre (sinon, on devrait proposer le dictionnaire complet comme correction possible).

Une fois le processus de correction lancé avec 'make_correction', plusieurs fonctions se succèdent pour réaliser cette correction. Mais tout d'abord, quelques explications sur cette fonction 'make_correction' :

- On crée et initialise dans un premier temps une liste contigüe de correction qui nous servira à contenir les mots possibles.
- On récupère un éventuel caractère de fin ('.', ',', '?', etc..) pour l'ajouter à la fin du mot de correction choisi.
- On coupe le mot avant l'erreur trouvée et on lance la fonction 'make_tree_correct'.

Quelques explications sur la fonction 'make_tree_correct' sont nécessaires avant de passer à la suite. Cette fonction va récupérer le pointeur d'arbre préfix juste avant l'erreur trouvée afin de chercher dans l'arbre toutes les fins de mot possibles. Pour cela, on procède de cette manière :

- On regarde tout d'abord si le pointeur est vide, auquel cas on sortira de la fonction car il n'y a plus (ou pas) de fin de mots disponibles.
- La fonction prend en paramètre une chaine de caractère (une chaine vide au lancement de la fonction). Cette chaine va être incrémentée d'une lettre si la cellule (donc le pointeur) n'est pas vide.
- Deux cas pour l'ajout à la liste contigüe : on arrive à une feuille ou on détecte un booléen de fin de mot à 'true'.
  Dans le premier cas, si on arrive à une feuille, c'est qu'il n'y a plus de mot ayant ce préfix, donc on peut imprimer le mot et supprimer la dernière lettre du mot qui ne mène à aucun autre.
  Dans le second cas, on imprime juste le mot car il peut éventuellement avoir d'autres mots ayant ce mot comme préfixe.
- Ensuite vient la partie la plus importante de la fonction. La fonction réalise une boucle sur le tableau de pointeurs de la cellule dans laquelle on se trouve, et si le pointeur n'est pas NULL, on relance la fonction (récursive du coup) à partir de la case pointée.
  Pour être plus clair, on parcourt le tableau de chaque cellule en descendant dans les tableaux des lettres suivantes si elles existent.
  A chaque cellule, on applique la totalité de la fonction pour stocker la lettre et effectuer les éventuels ajouts de mot par la fonction 'add_in_liste' dans la liste contigüe.
- Pour finir, on supprime la dernière lettre stockée car on aura fait le tour du tableau, donc celle-ci ne mènera à aucun autre mot : on retourne à l'étage précédent.

La fonction 'add_in_liste' prend en paramètre la liste et 2 chaines de caractère : le début du mot, et une fin possible. Elle concatène les deux avant de les insérer dans la liste.

Retournons dans la fonction 'make_correction'. Maintenant que nous avons la liste contigüe de tous les mots possibles pour le remplacement, il faut que l'utilisateur fasse un choix et que le processus de correction s'exécute. C'est ce que fait la fonction 'choice_word' qui demande à l'utilisateur s'il veut corriger le mot, et si oui, avec quel mot. La fonction va également ajouter l'éventuel caractère de fin détecté précédemment pour obtenir la bonne ponctuation. La fonction de correction ('correct_word') est ensuite lancée.

La fonction 'correct_word' va permettre au programme de remplacer le mot contenant une erreur par le mot choisi par l'utilisateur. Voici les étapes de l'algorithme :

- Tout d'abord, on crée un fichier tampon qu'on ouvre en mode lecture/écriture avec suppression du contenu au préalable.
- On va alors copier le début du fichier à analyser (jusqu'au mot avec erreur exclu) dans le fichier tampon.
- Ensuite, on y ajoute le mot de correction.
- Enfin, à l'aide des fonctions 'ftell' et 'fseek', on copie la fin du fichier à analyser à partir du mot de correction exclu dans le fichier tampon.
- A l'aide de la fonction 'rewind' et des fonctions de lecture et écriture dans les fichiers texte, on copie intégralement le fichier tampon dans le fichier original.
- Il suffit de s'aider des fonction 'ftell' et 'fseek' pour se placer au bon endroit dans le fichier à analyser, afin de reprendre l'analyse là où elle en était.

Ce processus est effectué pour chaque mot analysé (dans le cas où le mot est correct, la partie correction n'est bien sûr, pas lancée). Une fois le fichier complet analysé, la fonction 'main' va afficher le nombre d'erreurs présentes dans le fichier, et le nombre de mots qui ont été corrigés. Puis, elle va libérer l'espace mémoire alloué à l'arbre préfix.

IV. Test de performance

Etant donné que notre programme se met en attente d'une réponse de l'utilisateur s'il y a une erreur, on fera les tests de performance avec un texte sans erreur.

On considère le dictionnaire words contenant 99154 mots et le texte texte.txt contenant 294 mots. En moyenne, le temps d'exécution (à l'aide de la fonction time), est de :

- Real : 0m0.185s
- User : 0m0.105s
- Sys : 0m0.060s

V. Limites de notre projet

Notre projet n'est pas parfait, il comporte quelques limites que nous allons vous exposer ci-dessous.

- Tout d'abord, la gestion des accents. Nous avons choisi de ne pas les prendre en compte dans notre projet car nous récupérons le code ASCII des caractères (stockés dans des char) pour déterminer leur indice de stockage. En utilisant des char, il nous semblait compliqué de déterminé l'indice de rangement de ces caractères accentués. Nous avons préféré nous concentrer sur la partie correction que nous avons jugé plus intéressante.

- Ensuite, nous avons considérer le projet comme un correcteur orthographique et non grammatical. De ce fait, si 2 mots se suivent en étant séparer d'une virgule (par exemple) mais sans espace, le programme ne pourra différencier les 2 mots à cause de la fonction 'fscanf' qui coupe les mots au niveau des espaces.
  Nous aurions pu utiliser la fonction 'fgetc' pour limiter ce problème, mais ce genre de cas étant rare et n'étant pas un problème d'orthographe, nous avons préféré le laisser de côté.

- Enfin, au niveau de la correction, notre programme a ses limites.
  En effet, si on mot ne commence pas par une lettre reconnue par l'arbre, alors celui-ci ne peut être corrigé. Notre programme n'est pas capable d'analyser seulement la fin des mots et d'en proposer un début correct.
  Cependant, la correction des mots est opérationnelle, et son effet nous semble correct au vu du temps que nous avions pour réaliser ce projet, en parallèles d'autres projets et cours.