

Projet Dictionnaire

[Introduction](#)

[Vue d'ensemble](#)

[Solutions retenues](#)

[Restrictions liés au projet](#)

[Packages](#)

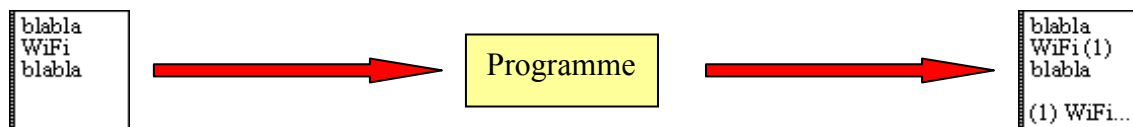
[Problèmes rencontrés](#)

[Améliorations possibles](#)

Introduction

Beaucoup de documents comprennent des mots difficiles à comprendre ou inconnus pour des novices dans le domaine concerné par le texte. Le programme développé va permettre de modifier un article en lui ajoutant des définitions de ces mots. Ainsi l'utilisateur aura un nouvel article compréhensible pour lui.

Le langage de programmation utilisé est ADA.



Vue d'ensemble

```
C:\ADA\main.exe

Projet Dico!

-> L'index a ete cree.

Menu
1 - Charger un index.
2 - Traiter un texte.
3 - Lire une definition.
4 - Ajouter une definition.
0 - Quitter!

Votre Choix : _
```

Solutions retenues

- Arbre binaire équilibré contenant comme informations mot & nom dico pour l'index.
- Dico.ini pour infos utiles (dernier fichier dico et nombre de définition contenues dans ce dernier)

- Une File (comme structure de données) permettant de mettre de mettre un (n°) à côté d'un mot compliqué et de reporter la définition à la fin.
- Le fichier traité a le même nom avec en plus un underscore « _ » avant son extension.
- Le programme gère pour l'instant tout type de fichier respectant le formatage des fichiers *.txt et html : *.htm & *.html.
- L'affichage se fait en mode console avec des couleurs.
- Gère les accents et les majuscules en ce qui concerne la reconnaissance de mots.

Restrictions liés au projet

- Un fichier dico à certaines restrictions :
 - le nom est sur 10 caractères, extension comprise.
 - nom spécifique et ordonné : 000001.dic, 0000002.dic, etc.
 - 10 définitions par fichiers avec une par ligne.
 - typage de la ligne : [mot] [espace] [:] [espace] [définition]
 - ligne doit être par conséquent inférieur à 1024 caractères.
 - définition soit être inférieure ou égale à 768 caractères.
- Un fichier index doit avoir un formatage de texte spécifique :
 - 1 ère ligne : nombre de mots
 - n ème lignes : [nom du mot] [espace] [:] [espace] [nom du dico].
- Tout type de fichier respectant le formatage de texte des « txt » peut être traité.
- Tout type de fichier respectant le formatage de texte des « html » peut être traité.
- Nom du nouveau fichier : article.txt => article_.txt, article => article_

Packages et fichiers utilisés

Text_IO ; Simple_IO ; NT_Console ;
 Gestion_Mots ; Gestion_Definitions ; Gestion_Dictionnaire ; Gestion_File ; Gestion_Index ;
 Main

Détails de certains packages

Gestion_Mots

```
-- Retourne un mot à partir d'une chaîne de caractères en entrée.
function Creer_Mot(S : in String) return pMot;

-- Retourne la longueur d'un mot.
function Lg_Mot(M : in pMot) return Natural;

-- Retourne le nième caractère d'un mot.
function NiemeChar_Mot(M : in pMot ; P : in Natural) return Character;

-- Retourne Vrai si la lettre a été validée.
function Valider_Lettre(C : in Character) return Boolean;

-- Compare deux mots de type pMot (-1, 0, 1)
function Compare_Mot(M1, M2 : in pMot) return Integer;
```

```
-- Enlève les majuscules, les accents, ... d'un mot de type pMot
function Basic_Mot(M : in pMot) return pMot;
```

Gestion_Definitions

```
-- Retourne une définition à partir d'une chaîne de caractère en entrée.
function Creer_Definition(S : in String) return pDefinition;
```

```
-- Retourne la longueur d'une définition.
function Lg_Definition(Def : in pDefinition) return Natural;
```

```
-- Retourne le nième caractère d'une définition.
function NiemeChar_Definition(Def : in pDefinition ; P : in Natural) return
Character;
```

Gestion_Dictionnaire

```
-- Ouvre un dictionnaire en lecture.
function Ouvrir_Lect_Dico(M : in pMot ; Fichier : in String) return
pDictionnaire;
```

```
-- Ferme un dictionnaire.
procedure Fermer_Dico(Dico : in pDictionnaire);
```

```
-- Retourne la définition d'un dico déjà ouvert.
function Lire_Def_Dico(Dico : in pDictionnaire) return pDefinition;
```

```
-- Crée/Ecrit/Ajoute une entité mot+définition dans un dictionnaire.
procedure Creer_Dico(M : in pMot ; Ind : out Index ; Nom : in String);
```

Gestion_File

```
-- Création de la file (FIFO).
procedure creer(F : out File);
```

```
-- Ajoute Mot et son numéro à la fin de la file (FIFO).
procedure empiler(F : out File ; N : in Natural ; E : in pMot);
```

```
-- Détruit le premier élément(n°&pMot) de la file (FIFO).
procedure depiler(F : in out File);
```

```
-- Renvoie True si la File (FIFO) est vide.
function vide(F : in File) return boolean;
```

```
-- Recherche un mot(pMot) dans la file (FIFO) et renvoie -1 si le mot
-- n'y est pas, son numéro dans le cas contraire.
function Recherche(F : in File ; M : in pMot) return Integer;
```

```
-- Ecrit dans le fichier le mot et la définition de tous les mots de la
-- file (FIFO).
procedure Afficher_Txt(F : in File ; Fichier : in File_Type ; Ind : in
Index);
```

```
-- Ecrit dans le fichier le mot et la définition de tous les mots de la
file (FIFO).
procedure Afficher_Html(F : in File ; Fichier : in File_Type ; Ind : in
Index);
```

Gestion_Index

```
-- Crée un index(arbre binaire) à partir d'un fichier contenant des mots
-- triés : méthode de parcours dichotomique.
function Creer_Index(Nom : in String) return Index;
procedure Insérer_Index(Fichier : in out File_Type ; Nom : in String ; Ind
: out Index ; Debut, Fin : in Natural);

-- Vérifie si un mot est dans l'index et retourne une chaîne de caractère
-- vide ou un nom de fichier dictionnaire (.dic)
function Appartient_Index(Ind : in Index ; M : in pMot) return String;

-- Parcours l'arbre pour pouvoir insérer un mot dans le fichier index
procedure Parcours_Index(Ind : in Index ; Fichier : in File_Type);

-- Affiche sous forme de liste ordonnée l'index(Arbre)
procedure Affiche_Index(Ind : in Index);

-- Affiche sous forme de graphique l'index(Arbre)
-- p:profondeur (0 en général)
procedure graphe_index(Ind : in Index ; p : in natural);

-- Ajoute un élément dans l'arbre tout en laissant celui-ci
-- équilibré (AVL) ; méthode de la hauteur et des rotations.
-- équilibrage simplifié !
procedure Ajout_Index(Ind : in out Index; h : in out boolean ; M : in pMot;
SDic : in String);
```

Main

```
-- Permet de traiter un fichier .txt !
procedure Traiter_Texte(Ind : Index ; Fil : in out File ; Fichier_Entree,
Fichier_Sortie : String);

-- Permet de traiter un fichier html !
procedure Traiter_HTML(Ind : Index ; Fil : in out File ; Fichier_Entree,
Fichier_Sortie : String);

-- Permet de lire une chaîne de caractère saisie par l'utilisateur.
procedure Lire_chaine(S : out String);

-- Va permettre d'enlever les espaces superflus à la fin d'une chaîne de
-- caractères.
function Traiter_Espaces(S : in String) return Natural;

-- Affiche le titre et le menu.
procedure Afficher_Projet ;

-- Assistance pour charger l'index.
procedure Menu_Charger_Index(S : String);
```

Problèmes rencontrés

- Màj du fichier index : peut pas insérer ligne sans utiliser in out => recréation fichier index en entier en utilisant l'arbre. => Temps nécessaire si le nombre de mots devient imposant. La solution consisterait en un index non classé avec un équilibrage d'arbre plus « sophistiqué ».
- Màj de l'arbre : équilibrage en temps réel de l'arbre simplifié car compliqué.

- Problème au niveau de l'affichage des accents dans la fenêtre dos.
- Essayer de gérer plus les exceptions... en cas d'erreurs.
- Faire des tests pour les fichiers html pour voir si des bogues apparaissent ! (ex : tableaux : formatage)

Améliorations possibles

- Gérer les pluriels, féminin, conjugaison... => au lieu d'un mot dans les données de l'arbre plusieurs (pile, file) => modification gestion de lecture des définitions & lorsque l'on cherche le mot. Meilleures solutions ?
- Tampon pour les expressions (2 mots ou plus) inconnues. Modifications mineures au niveau de la lecture d'un fichier dictionnaire.
- Gérer index principal et des sous index => permettrai nouvelles fonctionnalités (plus facile à gérer) comme supprimer mot index et personnalisation index en fonction du domaine (sujet) ;
- Pouvoir modifier des définitions.
- Dans fichier dic, en fonction de leur fréquence d'utilisation les placer dans les premières positions.
- Interface conviviale. (GTK). En plus permettrai de gérer les problèmes liés aux accents et symboles de la fenêtre dos ;)

Explications sur l'équilibrage des arbres binaires :

Un arbre binaire de recherche est efficace que s'il est bien équilibré. Un problème un peu difficile.

L'astuce est de s'appuyer sur les AVL : ce type d'arbre stocke une valeur supplémentaire d'équilibre. Ce nombre est égal à la hauteur du sous arbre du fils gauche du nœud moins la hauteur du sous arbre du fils droit du nœud. Au fur et à mesure que les éléments sont ajoutés dans l'arbre, il se rééquilibre. Toutes ses valeurs d'équilibres restent à -1, 0 ou +1. +1 signifie pesant à gauche ; -1 signifie pesant à droite ; et 0 équilibré. On doit donc, après avoir inséré un élément dans l'arbre, faire des modifications pour garder l'arbre équilibré grâce à des rotations. Une rotation permet de rééquilibrer une partie de l'arbre en réarrangeant les nœuds tout en respectant la propriété d'un arbre de recherche. (Rotations GG, GD, DD, DG)

Réflexion de solution basique pour les mots dérivés :

Dans les fichiers dico, séparer les mots dérivés par une virgule.

Ex : vendeur, vendeuse, vendeurs, vendeuses : personne qui échange de l'argent contre un bien quelconque.

Si l'utilisateur rentre un nouveau mot, lui demander s'il veut ajouter des mots dérivés (pluriels, conjugaison, féminin) associés.

Dans l'index, il faut au lieu d'un mot comme info, mettre une liste de mots.

Le problème est qu'il faut éviter de comparer tous les mots de la liste. Pour cela, dans la liste, il y a une case numéro, il faut y mettre le nombre de lettres communes (début des mots) à tous les mots dérivés. Ensuite, il faudrait modifier la fonction qui compare deux mots pour qu'elle renvoie 0 si les deux mots sont égaux, un nombre positif signifiant que le premier mot est supérieur au deuxième et sa valeur étant le nombre de lettres communes aux deux mots, négatif dans le cas contraire. Si le numéro du mot est égal à la valeur absolue de la valeur retournée par la fonction compare alors compare avec les autres éléments de la liste du nœud courant.

Mesure du temps des opérations pour améliorer certaines fonctions, observer les lenteurs :

```
With Text_IO;
with Ada.Real_Time; use Ada.Real_Time;

-- variables
temps0 : Time;
temps1 : Time;
elapsed : Time_Span;
package Duration_IO is new Text_IO.Fixed_IO (Duration);

-- dans le programme
temps0 := Clock;
-- opérations
temps1 := Clock;
elapsed := t1 - t0;

-- pour afficher le temps passé
-- To_Duration (TS : Time_Span) return Duration;
Duration_IO.Put (To_Duration (elapsed));

-- une pause
delay 0.0025;
```

ESTIENNE Sébastien