

Avrile Floro (n° étudiant : 22000086)

Algorithmique et structures de données 1

Le 26 décembre 2023



Table des matières

1	Listes 6 Dico	3
1.1	Réalisation du prototype en Lisp	3
1.1.1	Le choix du dictionnaire et le traitement préalable	3
1.1.2	La fonction lire-fichier	4
1.1.3	Le parenthésage facile	5
1.1.4	Détailler la forme à traduire et la traduction	8
1.1.5	La lecture des données	13
1.1.6	La reconstruction du dictionnaire	13
1.1.7	L'accès aux données	17
1.1.8	Le programme de traduction	23
1.2	Programme en C - Analyse des données	25
1.3	Programme en C - Présentation du programme	27
1.4	Programme en C - Le lancement des tests	45
1.5	Programme en C - Exemple d'utilisation du dictionnaire bilingue	49
	Annexes	52
.1	Le code source "prototype.lisp"	52
.2	Le code source "dictionnaire.c"	62
.3	L'intégralité du fichier "category.c"	68
.4	L'intégralité du fichier "test.c"	71
.5	L'intégralité du fichier "structure.h"	73
.6	L'intégralité du fichier "sys.h"	75
.7	L'intégralité du makefile	76
.8	L'intégralité du makefile_tests	77

Algorithmique et structures de données 1

Listes 6 Dico

Avrile Floro

Étudiante n°22000086

1 Listes 6 Dico

Choisissez l'un de ces dictionnaires (ou un autre du même genre) et produisez d'abord une analyse des données. Faites une maquette d'un programme de traduction en ligne en Lisp (ou en Python) ; économisez au maximum l'espace mémoire. Portez ensuite cette maquette en C, en utilisant une interface rudimentaire (scanf pour lire le mot à traduire, printf pour écrire la traduction).

Fournissez en PDF l'analyse des données et toutes vos observations sur le problème, ses difficultés, et sur la solution que vous avez élaborée. **Des rendus uniquement constitués de code, pour cet exercice et pour les suivants, ne seront pas acceptés.**

Le sujet du mail doit être : Listes 6 Dico.

1.1 Réalisation du prototype en Lisp

Nous avons décidé de réaliser le prototype du dictionnaire en Lisp. Lors de la rédaction de ces lignes, nous avons terminé l'ensemble du cours. Nous avons trouvé la réalisation du prototype en Lisp être la partie la plus difficile (pour nous) du cours. Avec le recul, nous pensons que nous aurions dû choisir Python qui semblait plus approprié pour cette tâche.

Nous avons eu de nombreuses difficultés, notamment le fait que nous avons initialement utilisé la récursion sur un extrait de quelques milliers de références. Néanmoins, lors du passage au dictionnaire complet, cela ne marchait plus. Nous avons dû réécrire tout le code de façon itérative...

Nous avons également eu l'impression d'être peu efficace dans la rédaction de ce code en Lisp (qui est encore plus long que le dictionnaire en C!). Néanmoins le code fonctionne.

Nous avons illustré notre programme au fur et à mesure avec l'utilisation de variables globales. Ces variables globales n'existent pas en tant que telles dans le programme original. Elles sont à la place déclarées comme des variables locales dans notre fonction `main` et participent à la création de notre dictionnaire.

1.1.1 Le choix du dictionnaire et le traitement préalable

Nous avons utilisé le dictionnaire anglais-espagnol issu du lien suivant <https://github.com/mananoreboton/en-es-en-Dic/blob/master/src/main/resources/dic/en-es.xml>.

Après traitement des données, nous avons 30 664 entrées dans notre dictionnaire. Le dictionnaire que nous avons choisi n'est pas idéal mais au moment du choix, les liens proposés par le cours n'étaient pas fonctionnels. Nous avons perdu énormément de temps (des heures!) à traiter les entrées erronées du dictionnaire. Nous choisirons avec plus de soin notre support à l'avenir car nous avons pu constater à quel point un support mal adapté pouvait être source de difficultés.

Nous détaillerons plus en détail notre dictionnaire dans la partie du rendu dédié à notre programme en C.

Les entrées de notre dictionnaire sont sous cette forme : `mot_anglais {cat_grammaticale} mot_espagnol1 {cat_gram_optionnelle}, mot_espagnol2.`

```

aberrant {adj} anormal, atípico, aberrante, anómalo
aberration {n} perturbación mental {f}, aberración {f}
abessive case {n} caso abesivo {m}
abet {v} ayudar, incitar
abettor {n} cómplice {m}{f}

```

FIGURE 1 – La présentation de notre dictionnaire anglais-espagnol.

1.1.2 La fonction lire-fichier

La fonction `lire-fichier` lit un fichier et divise son contenu en une liste de lignes.

La fonction ouvre le fichier qui lui est passé en argument et commence une boucle qui lit chaque ligne du fichier. Tant qu'il y a des lignes à lire dans le fichier, la fonction crée une liste contenant chaque ligne lue et l'ajoute à la liste principale.

Le résultat obtenu est une liste de listes, où chaque sous-liste contient une ligne du fichier.

```

1 ; Nom ..... : prototype.lisp
2 ; Role ..... : prototype du dico en lisp
3 ; Auteur ..... : AvriLe Floro
4 ; Version ..... : V0.1 du 22/12/23
5 ; Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 ; Usage : lisp puis (load "prototype.lisp") puis (traduction 'love '{v})
7
8 ; NAME : lire-fichier ; divise le dico en liste de lignes
9 ; ARGS : fichier en entrée
10 ; USAGES : (lire-fichier fichier)
11 ; GLOBALS : none
12 ; CALL : none
13 ; USER : main
14
15 (defun lire-fichier (nom_fichier)
16   (with-open-file (fichier nom_fichier)
17     (loop for ligne = (read-line fichier nil) ; lit une ligne
18       while ligne ; tant qu'il y a une ligne
19       collect (list ligne) ) ) ; chaque sous liste contient une ligne

```

Le résultat de la fonction `lire-fichier` peut être illustré par la création d'une variable globale `etape1`. Ceci est à but purement indicatif (cette variable n'est pas nécessaire au fonctionnement du programme).

```

21 ; NAME : etape1 ; le fichier sous forme de liste de listes
22 ; USAGES : appel de la fonction lire-fichier > (("aback {adv} hacia atrás")
23 ; ("another day, another dollar {proverb}
24 ; USER :
25
26 ; (setq etape1 (lire-fichier "mon_dico.txt"))

```

```
Break 2 [3]> (setq etape1 (lire-fichier "mon_dico.txt"))
(("a {art} uno {art}, una {art}") ("Aachen {prop} Aquisgrán")
 ("aah {interj} ¡ah!, ¡ay!") ("Aalenian {adj} aaleniano")
 ("Aalenian {prop} aaleniano {m}") ("aalii {n} circa {f}")
 ("aardvark {n} cerdo hormiguero {m}")
 ("aardwolf {n} proteles {m}, lobo de tierra {m}")
 ("Aaron {prop} Aarón") ("abaca {n} abacá {m}")
 ("abacist {n} abacista {m}{f}") ("aback {adv} hacia atrás")
 ("abacus {n} ábaco {m}") ("Abadan {prop} Abadán")
 ("abaft {adv} a popa") ("abaft {prep} a popa"))
```

FIGURE 2 – Définition de la variable `etape1` avec l'appel de la fonction `lire-fichier` prenant `mon_dico.txt` en argument. Nous définissons des variables globales uniquement pour illustrer nos fonctions.

1.1.3 Le parenthésage facile

Nous avons suivi les recommandations du cours pour le parenthésage avec :

- Une liste pour le dictionnaire (d'après la fonction `lire-fichier`)
- Une liste pour chaque entrée du dictionnaire (d'après la fonction `lire-fichier`)
- Deux sous-listes illustrant la division anglais/espagnol (d'après les fonctions `div-english` et `div-spanish`)
- Deux sous-listes au sein de la partie en anglais pour le mot en anglais et sa catégorie (d'après les fonctions `eng-word` et `cat-word`).

La création d'une liste pour le dictionnaire et d'une sous-liste pour chaque entrée a été faite grâce à la fonction `lire-fichier`. Pour isoler les différentes sous-parties de la chaîne, nous avons utilisé `position` et `subseq`.

La fonction `div-english`

La fonction `div-english` prend en entrée une chaîne de caractères et utilise un index pour localiser la fin de la partie en anglais. Elle supprime les espaces superflus et renvoie le résultat sous forme d'une liste.

```
28 ; NAME : div-english ; isole la partie anglaise de la ligne dans une liste
29 ; ARGS : chaine &aux index-fermeture english
30 ; USAGES : (div-english chaine) > ("aback {adv}")
31 ; GLOBALS : none
32 ; CALL : none
33 ; USER : parentheses
34
35 (defun div-english (chaine &aux index-fermeture english)
36   (setq index-fermeture (position #\} (car chaine))) ; cherche la position de la fermeture
37   (setq english (string-trim '(#\Space) (subseq (car chaine) 0 (1+ index-fermeture))))
38   ; isoler la partie anglaise
39   (list english) ) ; renvoie la partie anglaise dans une liste
```

La fonction `eng-word`

La fonction `eng-word` renvoie le mot anglais d'une chaîne de caractères qui contient à la fois le mot anglais

et sa catégorie grammaticale. La fonction identifie le début de la catégorie grammaticale et extrait le mot qui la précède. Le mot est nettoyé des espaces et est retourné dans une liste.

```
42 ; NAME : eng-word ; isole le mot anglais dans une liste
43 ;           depuis la catégorie anglais
44 ; ARGS : chaine &aux index-ouverture mot
45 ; USAGES : (eng-word chaine) > ("aback")
46 ; GLOBALS : none
47 ; CALL : none
48 ; USER : parentheses
49
50 (defun eng-word (chaine &aux index-ouverture mot)
51   (setq index-ouverture (position #\{ (car chaine))) ; cherche la position de l'ouverture
52   (setq mot (string-trim '#\Space) (subseq (car chaine) 0 index-ouverture)))
53   ; isole le mot anglais (avant l'ouverture)
54   (list mot) ) ; renvoie le mot anglais dans une liste
```

La fonction cat-word

La fonction `cat-word` isole et renvoie la catégorie grammaticale d'un mot anglais à partir d'une chaîne de caractères. La fonction utilise deux index pour localiser la catégorie grammaticale et extrait cette dernière. La catégorie est ensuite retournée dans une liste.

```
57 ; NAME : cat-word ; isole la catégorie du mot anglais
58 ;           dans une liste depuis la catégorie anglais
59 ; ARGS : chaine &aux index-ouverture index-fermeture mot
60 ; USAGES : (eng-word chaine) > ("adv")
61 ; GLOBALS : none
62 ; CALL : none
63 ; USER : parentheses
64
65 (defun cat-word (chaine &aux index-ouverture index-fermeture mot)
66   (setq index-ouverture (position #\{ (car chaine))) ; cherche la position de l'ouverture
67   (setq index-fermeture (position #\} (car chaine))) ; cherche la position de la fermeture
68   (setq mot (string-trim '#\Space) (subseq (car chaine) index-ouverture (1+ index-fermeture))))
69   ; isole la catégorie du mot anglais (entre les deux)
70   (list mot) ) ; renvoie la catégorie du mot anglais dans une liste
```

La fonction div-spanish

La fonction `div-spanish` extrait et renvoie la partie en espagnol d'une chaîne de caractères. La partie en espagnol se trouve après la catégorie grammaticale du mot anglais. La fonction trouve la fin de la partie anglaise à l'aide d'un index et isole la partie en espagnol qui suit. Cette portion est retournée sous forme d'une liste.

```
74 ; NAME : div-spanish ; isole la partie espagnole de la ligne dans une liste
75 ; ARGS : chaine &aux index-fermeture spanish
76 ; USAGES : (eng-word chaine) > ("hacia atrás")
77 ; GLOBALS : none
78 ; CALL : none
79 ; USER : parentheses
80
81 (defun div-spanish (chaine &aux index-fermeture spanish)
82   (setq index-fermeture (position #\} (car chaine))) ; cherche la position de la fermeture
83   (setq spanish (string-trim '#\Space) (subseq (car chaine) (1+ index-fermeture))))
```

```

84 ; isole la partie espagnole (après la fermeture)
85 (list spanish) ) ; renvoie la partie espagnole dans une liste

```

La fonction parentheses

Finalement, l'ensemble des fonctions intermédiaires peuvent être appelées par la fonction **parentheses** qui boucle sur l'ensemble du dictionnaire de façon itérative.

La fonction prend une liste d'éléments en entrée et traite chaque élément individuellement. Elle extrait la partie en anglais avec **div-english**, le mot anglais avec **eng-word**, la catégorie grammaticale du mot anglais avec **cat-word**, et la partie espagnole avec **div-spanish**.

Nous avons utilisé la fonction **push** et **nreverse** plutôt que **append** pour améliorer les performances.

```

87 ; NAME : parentheses ; fais le parenthésage facile (grâce aux sous fonctions)
88 ; ARGS : liste &aux resultat engpart engmot catmot spanish ligne
89 ; USAGES : (parentheses etape1) > (((("aback") ("adv")) ("hacia atrás")))
90 ; (((("another day, another dollar") ("proverb")) ("un día más un día menos")))
91 ; ("hyper-")
92 ; GLOBALS : none
93 ; CALL : div-english, eng-word, cat-word, div-spanish
94 ; USER : main
95
96 (defun parentheses (liste &aux resultat engpart engmot catmot spanish ligne)
97   (loop for item in liste do
98     (setq engpart (div-english item)) ; partie anglaise
99     (setq engmot (eng-word engpart)) ; mot anglais
100    (setq catmot (cat-word engpart)) ; catégorie du mot anglais
101    (setq spanish (div-spanish item)) ; partie espagnole
102    (setq ligne (list (list engmot catmot) spanish)) ; rassemble le tout dans la ligne
103    (push ligne resultat) )
104    (nreverse resultat) )

```

Illustration du fonctionnement de la fonction parentheses

Pour illustrer notre fonction, on définit une nouvelle variable, **etape2** qui correspond à l'appel de la fonction **parentheses** avec le passage de **etape1** en argument.

Encore une fois, ces variables globales sont purement illustratives et ne sont pas nécessaires au bon fonctionnement du programme.

```

109 ; NAME : etape2 ; on applique à etape1 le parenthésage facile
110 ; USAGES : appel de la f. parentheses > (((("aback") ("adv")) ("hacia atrás"));
111 ; (((("another day, another dollar") ("proverb")) ("un día más un día menos")))
112 ; ("hyper-")
113 ; USER :
114 ; (setq etape2 (parentheses etape1))

```

À l'issue de cette étape, chacune des lignes de mon dictionnaire est structurée ainsi :

```

(((anglais) ({catégorie})) (espagnol))

```



```
Break 2 [3]> (setq etape2 (parentheses etape1))
((((("a") ("{art}")) ("uno {art}, una {art}"))
  (((("Aachen") ("{prop}")) ("Aquisgrán"))
    (((("aah") ("{interj}")) ("¡ah!, ¡ay!"))
      (((("Aalenian") ("{adj}")) ("aaleniano"))
        (((("Aalenian") ("{prop}")) ("aaleniano {m}"))
          (((("aalii") ("{n}")) ("chirca {f}"))
            (((("aardvark") ("{n}")) ("cerdo hormiguero {m}"))
              (((("aardwolf") ("{n}")) ("proteles {m}, lobo de tierra {m}"))
                (((("Aaron") ("{prop}")) ("Aarón"))
                  (((("abaca") ("{n}")) ("abacá {m}"))
```

FIGURE 3 – La variable `etape2` correspond à l'appel de la fonction `parentheses` en lui passant `etape1` en argument.

1.1.4 Détailler la forme à traduire et la traduction

La structure de la forme à traduire dans notre dictionnaire est :

```
mot {catégorie}
```

Nous avons donc décidé de créer un doublet pour y mettre le mot et sa catégorie. Pour ce faire, nous utiliserons la fonction `cons` avec `(caaar (car etape2))` et `(caadar (car etape2))` au sein de la fonction `structure_en_cours` que nous allons détailler ci-après.

La fonction `split-by-one-comma`

Les différentes traductions des mots dans notre dictionnaire sont séparées par des virgules. Il n'existe pas de fonction native en Lisp permettant la séparation des chaînes par un caractère.

Nous avons trouvé sur <https://cl-cookbook.sourceforge.net/strings.html> une fonction permettant la division d'une chaîne par un caractère. Nous avons adapté cette fonction afin qu'elle sépare notre chaîne au niveau des virgules. Cette fonction nous a été très utile et nous l'avons notamment réutilisée plus loin dans le cours pour le système expert.

```
117 ; NAME : split-by-one-comma ; sépare une chaîne de caractères en deux sous liste
118 ;           à la virgule utiliser pour séparer les différentes
119 ;           traductions possibles
120 ; ARGS : string
121 ; USAGES : (split-by-one-comma "a, v") > ("a" " v")
122 ; GLOBALS : none
123 ; CALL :
124 ; USER : isoler_diff_traduc
125 ; SOURCE: https://cl-cookbook.sourceforge.net/strings.html
126
127 * (defun split-by-one-comma (string)
128   (loop for i = 0 then (1+ j)
129     as j = (position #\, string :start i)
130     collect (subseq string i j)
131     while j) )
```

La fonction `isoler_diff_traduc`

Nous avons créé une fonction `isoler_diff_traduc` qui appelle `split-by-one-comma` sur le `caadr` de la

liste, ce qui correspond à la partie en espagnol.

Ainsi, la fonction sépare les différentes traductions espagnoles qui sont regroupées dans une seule chaîne.

```
133 ; NAME : isoler_diff_traduc ; appelle split-by-one-comma sur la partie espagnole de la liste
134 ;     donne une sous liste pour chaque traduction possible
135 ; ARGS : liste (avec le full parenthésage)
136 ; USAGES : (isoler_diff_traduc (L)) > ("hacia atrás {v}" " hacia detrás {adv}")
137 ; GLOBALS : none
138 ; CALL :
139 ; USER : structure_en_cours
140
141 (defun isoler_diff_traduc (L)
142   (if (atom L) ; si fin de la liste
143       nil ; nil
144       (split-by-one-comma (caadr L)) ) )
145   ; sinon on appelle split-by-one-comma sur la partie espagnole de la liste
```

La fonction sous_liste

La fonction `sous_liste` crée une nouvelle liste où chaque élément de la liste d'origine devient une sous-liste.

```
147 ; NAME : sous_liste ; crée des sous_liste à partir des éléments de la liste
148 ; ARGS : L &aux result
149 ; USAGES : (sous_liste '(a b c)) > ((a) (b) (c))
150 ; GLOBALS : none
151 ; CALL :
152 ; USER : structure_en_cours
153
154 (defun sous_liste (L &aux result)
155   (if (atom L)
156       nil ; Si L est un atome, renvoie nil
157       (loop for item in L do (push (list item) result)) )
158   ; Pour chaque élément de L, ajoute une sous-liste à result
159   (nreverse result) ) ; Renvoie result en ordre inversé
```

La fonction structure_en_cours

La fonction `structure_en_cours` place chaque mot anglais et sa catégorie dans un doublet, et chaque traduction espagnole possible est placée dans une sous-liste séparée.

La fonction utilise `isoler_diff_traduc` pour séparer les différentes traductions espagnoles en sous-listes. Chaque entrée est composée d'un doublet contenant le mot anglais et sa catégorie, et d'une liste des sous-listes de traductions espagnoles.

```
163 ; NAME : structure_en_cours ; crée un doublet pour le mot anglais et sa catégorie
164 ;     et crée des sous-listes pour chaque traduction possible >> il faudra
165 ;     encore isoler les catégories de l'espagnol
166 ; ARGS : liste &aux result prep new-item
167 ; USAGES : (sous_liste '(a b c)) > ((a) (b) (c))
168 ; GLOBALS : none
169 ; CALL : sous_liste, isoler_diff_traduc
170 ; USER : main
171
172 (defun structure_en_cours (liste &aux result prep new-item)
```

```

173 (dolist (item (reverse liste) result) ; pour chaque élément de la liste
174   (setq prep (isoler_diff_traduc item))
175   ; on sépare en sous-listes les différentes traductions du mot
176   (setq new-item (cons (cons (caaar item) (caadar item)) (list (sous_liste prep)))))
177   ; on crée un doublet pour le mot anglais et sa catégorie
178   ; on crée des sous-listes pour chaque traduction possible
179   (push new-item result) ) ; on ajoute à la liste résultat

```

Illustration avec la variable globale etape3

Afin d'illustrer le fonctionnement de notre programme, nous déclarons la variable **etape3** qui correspond à l'appel de la fonction **structure_en_cours** avec **etape2** en argument. Cette variable globale n'existe pas comme telle dans notre programme mais nous l'utilisons à des fins d'illustration.

```

182 ; NAME : etape3 ; on applique à etape2 la construction en doublet pour l'anglais
183 ;          et les sous-liste de traduction pour l'espagnol
184 ; USAGES : (("awful" . "{adj}") ("horrible {adj}") ("espantoso {adj}") ("terrible {adj}")
185 ;           ("tremendo {adj}"))
186 ; USER :
187 ; (setq etape3 (structure_en_cours etape2))

```

```

[4]> (setq etape3 (structure_en_cours etape2))
(((("a" . "{art}") ("uno {art}") ("una {art}"))
  ("Aachen" . "{prop}") ("Aquisgrán"))
  ("aah" . "{interj}") ("¡ah!" ("¡ay!")))
  ("Aalenian" . "{adj}") ("aaleniano"))
  ("Aalenian" . "{prop}") ("aaleniano {m}"))
  ("aalii" . "{n}") ("chirca {f}"))
  ("aardvark" . "{n}") ("cerdo hormiguero {m}")))

```

FIGURE 4 – La variable **etape3** correspond à l'appel de la fonction **structure_en_cours** en lui passant **etape2** en argument.

À l'issue de la création de la variable **etape3** (qui est une étape intermédiaire), nous avons la partie traduction présentée comme suit (seule la partie traduction est indiquée) :

```

(("oreja de mar {f}") ("abulón {m}") ("loco {m}"))
(("desenfreno {m}"))
(("renunciar") ("abandonar"))

```

Comme nous pouvons le constater, la catégorie n'est pas toujours présente. Nous devons donc maintenant séparer la traduction et sa catégorie, et ajouter la catégorie du mot anglais si la traduction espagnole ne présente pas de catégorie.

Nous avons décidé de ne pas créer de sous-catégorie pour les noms car certaines traductions ne présentant pas de catégorie vont hériter de la catégorie générale **{n}** de l'anglais. Nous laisserons donc les catégories de noms simples et distinctes avec, respectivement **{n}**, **{f}**, **{m}**, **{f.p}**, **{m.p}**, etc.

Certains mots ont été catégorisés comme pluriel mais cela n'est pas généralisé ni généralisable à tous les mots. Ces distinctions ne s'appliquent qu'à l'espagnol car pour l'anglais, une seule catégorie **{n}** est utilisée.

La fonction process-sublist

La fonction `process-sublist` divise une sous-liste en deux éléments : la traduction espagnole et sa catégorie, puis les assemble en un doublet.

La fonction cherche la position des caractères indiquant une catégorie grammaticale dans la sous-liste contenant la traduction espagnole. Si aucune traduction n'est disponible, alors la fonction utilise celle du mot anglais. Le tout est renvoyé sous forme de doublet.

La fonction `process-sublist` a deux arguments `liste` et `liste1bis` qui correspondent à la même liste, néanmoins à des profondeurs différentes afin de pouvoir récupérer la catégorie du mot anglais. Nous avons eu des difficultés à obtenir un résultat satisfaisant sur cette fonction. La différence de profondeur a rendu la tâche difficile.

```
190 ; NAME : process-sublist ; divise une sous liste en > la traduction espagnole et sa catégorie
191 ;                               et construit les doublets
192 ; ARGS : liste liste1bis &aux index-ouverture index-fermeture cat
193 ; USAGES : (("llorón {m}") (" quejumbroso {m}")) > ("llorón" . "{m}")
194 ; GLOBALS : none
195 ; CALL :
196 ; USER : process-list
197
198 (defun process-sublist (liste liste1bis &aux index-ouverture index-fermeture cat)
199   (if (position #\{ (car liste)) ; S'il y a une catégorie pour le mot espagnol
200       (progn ; on retournera le résultat du cons
201           (setq index-ouverture (position #\{ (car liste))) ; Définit l'ouverture
202           (setq index-fermeture (position #\} (car liste))) ; Définit la fermeture
203           (setq cat (string-trim '(#\Space) (subseq (car liste) index-ouverture (1+ index-fermeture))))
204           ; Isole la catégorie de la traduction espagnole
205           (cons (string-trim '(#\Space) (subseq (car liste) 0 index-ouverture)) cat) )
206       ; Met sur un doublet le mot et sa catégorie
207       (cons (string-trim '(#\Space) (subseq (car liste) 0)) (cdr liste1bis)) ) )
208 ; Sinon, doublet avec mot espagnol et catégorie issue de l'anglais via liste1bis
```

La fonction process-list

La fonction `process-sublist` ne permet l'ajout de la catégorie que pour une seule sous-liste. Afin de traiter l'ensemble des sous-listes d'une traduction, on a créé une autre fonction qui s'appelle `process-list`.

La fonction `process-list` applique la fonction `process-sublist` à chaque sous-liste des traductions espagnoles pour construire une liste de doublets.

La fonction parcourt une liste de sous-listes et applique `process-sublist` à chacune.

```
213 ; NAME : process-list ; applique process-sublist à une liste de sous-liste
214 ; ARGS : liste liste1bis &aux (resultat nil)
215 ; USAGES : (("llorón {m}") (" quejumbroso {m}")) > (("llorón" . "{m}") ("quejumbroso" . "{m}"))
216 ; GLOBALS : none
217 ; CALL : process-sublist
218 ; USER : forme_finale
219
220 (defun process-list (liste liste1bis &aux resultat)
221   (loop for element in liste do ; pour chaque elt
222       (push (process-sublist element (car liste1bis)) resultat) )
223   ; on ajoute le résultat de process-sublist à la liste résultat
224   (nreverse resultat) ) ; on renvoie la liste résultat inversée
```

La fonction forme-finale

La fonction `forme_finale` construit la structure finale en associant chaque mot anglais et sa catégorie avec la liste de ses traductions espagnoles et leurs catégories respectives.

La difficulté pour ces fonctions a été la gestion des différents niveaux de profondeur des listes et aussi la gestion de la liste permettant d'associer la catégorie de l'anglais à la traduction espagnole.

```
245 ; NAME : forme_finale ; finit la construction de la liste en doublet pour l'anglais et pour l'espagnol
246 ; ARGS : liste liste1bis &aux res
247 ; USAGES : (forme_finale etape3 etape3)
248 ; GLOBALS :
249 ; CALL : process-list
250 ; USER : main
251
252 (defun forme_finale (liste liste1bis &aux res)
253   (dolist (elt liste res) ; pour chaque élément de la liste
254     (setq res
255       (cons ; on ajoute le mot traité à la liste résultat
256         (cons ; pour l'anglais + pour l'espagnol
257           (car elt)
258           (list (reverse (process-list (cadr elt) (car liste1bis))))))
259       res) )
260     (setq liste1bis (cdr liste1bis)) ) ; on itère sur la suite de liste1bis
261   (reverse res) ) ; on renvoie la liste résultat inversée
```

Exemple d'utilisation de la fonction `forme_finale` avec la variable `etape4`

À des fins purement illustratives (notre programme n'a pas besoin de ces variables globales pour fonctionner), nous déclarons une variable `etape4` qui correspond à l'appel de la fonction `forme_finale` avec la variable `etape3`.

```
250 ; NAME : etape4 ; on applique à etape3 la construction en doublet pour les sous listes de l'espagnol
251 ; USAGES : (("bellyacher" . "{n}") (("llorón" . "{m}") ("quejumbroso" . "{m}")))
252 ; USER :
253 ; (setq etape4 (forme_finale etape3 etape3))
```

```
]> (setq etape4 (forme_finale etape3 etape3))
("a" . "{art}") (("una" . "{art}") ("uno" . "{art}"))
("Aachen" . "{prop}") (("Aquisgrán" . "{prop}"))
("aah" . "{interj}") ((";ay!" . "{interj}") ("¡ah!" . "{interj}"))
("Aalenian" . "{adj}") (("aaleniano" . "{adj}"))
("Aalenian" . "{prop}") (("aaleniano" . "{m}"))
("aalii" . "{n}") (("chirca" . "{f}"))
("aardvark" . "{n}") (("cerdo hormiguero" . "{m}"))
("aardwolf" . "{n}")
(("lobo de tierra" . "{m}") ("proteles" . "{m}"))
```

FIGURE 5 – La variable `etape4` correspond à l'appel de la fonction `forme_finale` en lui passant `etape3` en argument.

Toutes les données sont maintenant organisées et traitées.

1.1.5 La lecture des données

Notre structure de données de base pour l'entrée du dictionnaire est : (la première parenthèse correspond à l'ouverture de la liste du dictionnaire)

```
((mot_anglais . catégorie) ((traduction_espagnole . catégorie) (traduction_espagnole . catégorie)))  
((mot_anglais . catégorie) ((traduction_espagnole . catégorie)))
```

La fonction main

Les étapes successives présentées précédemment avec des variables globales (à une fin d'illustration) sont réalisées automatiquement grâce à la fonction `main` qui est exécutée au chargement du fichier Lisp. Nous allons présenter dans la suite les tests qui sont également réalisés automatiquement avec le chargement de cette fonction.

La fonction `main` dirige le traitement du dictionnaire depuis la lecture initiale du fichier jusqu'à la reconstruction finale pour vérification.

Cette fonction commence par la lecture du fichier (`lire-fichier`), l'ajout de structure (`parentheses`), la séparation des traductions (`structure_en_cours`), et la construction finale (`forme_finale`). Elle s'occupe également de reconstituer notre dictionnaire (`printe`) et de l'imprimer dans un fichier (`imprimer`). La fonction appelle également la fonction `test` qui imprime dans divers fichiers les tests effectués (`test`).

```
526 ; NAME : main ; la fonction principale realise les traitements successifs  
527 ; ARGS : entree sortie_reconstruction &aux etape1 etape2 etape3 etape4 myprint  
528 ; USAGES : (main "mon_dico.txt" "reconstruction_dico.txt") > nil  
529 ; GLOBALS :  
530 ; CALL : lire-fichier, parentheses, defcat, structure_en_cours, printe, imprimer, test  
531 ; USER : toplevel  
532  
533 (defun main (entree sortie_reconstruction &aux etape1 etape2 etape3 etape4 myprint)  
534   (setq etape1 (lire-fichier entree)) ; lit le fichier ligne par ligne  
535   (setq etape2 (parentheses etape1)) ; ajoute les parenthèses  
536   (setq etape3 (structure_en_cours etape2)) ; isole chaque différente traduction possible  
537   (setq etape4 (forme_finale etape3 etape3))  
538   ; ajoute les catégories aux différentes traductions  
539   (setq myprint (printe etape2))  
540   ; reconstruit la liste de départ à partir de l'étape 2 (avant modification du contenu du dictionnaire)  
541   (imprimer myprint sortie_reconstruction)  
542   ; imprime le résultat de notre reconstruction en sortie  
543   (test etape4) ) ; pour vérifier les accès aux données
```

```
547 ; NAME : appelle main pour réaliser les traitements sur fichier test et dico original  
548 ; USAGES : vérification bon fonctionnement programme complet  
549 ; USER :  
550  
551 (main "mon_dico.txt" "reconstruction_dico.txt")
```

1.1.6 La reconstruction du dictionnaire

Pour reconstruire notre dictionnaire, nous allons partir de `etape2` (qui n'est pas une variable globale mais bien une variable locale déclarée dans `main`). En effet, à partir de l'étape3 nous avons altéré le contenu du dictionnaire en supprimant les virgules. Contrairement au programme écrit en C, nous n'avons pas gardé trace

de la forme originale dans notre prototype.

Nous avons écrit des fonctions permettant d'isoler pour chaque sous-liste d'étape2 : le mot anglais, la catégorie du mot anglais et la partie espagnole (avant son traitement).

```
→ prototype_LISP diff /Users/avri/le/Desktop/Algo1/6.\ Exo\ Listes\ 6\ Dico/6.\ Exo\ Listes\ 6\ Dico/
prototype_LISP/mon_dico.txt /Users/avri/le/Desktop/Algo1/6.\ Exo\ Listes\ 6\ Dico/6.\ Exo\ Listes\ 6\
Dico/prototype_LISP/reconstruction_dico.txt
30644c30644
< zygote {n} cigoto {m}
\ No newline at end of file
---
> zygote {n} cigoto {m}
→ prototype_LISP
```

FIGURE 6 – Le document reconstruit par notre fonction est effectivement identique au dictionnaire original.

La fonction imprimer-anglais

La fonction `imprimer-anglais` extrait et renvoie uniquement les mots anglais à partir de la structure de données `étape2`.

La fonction utilise `mapcar` pour appliquer `caaar` à chaque élément de la liste, ce qui extrait la première partie de chaque sous-liste, correspondant au mot anglais.

```
256 ; NAME : imprimer-anglais ; renvoie que les mots en anglais (on utilise etape2
257 ; car à partir de etape3 on a modifié la construction du dico > en vue des comparaisons)
258 ; ARGS : liste
259 ; USAGES : (imprimer-anglais etape2) ("aback" "another day, another dollar" "hyper-" "-ic" "Majesty"
      "Mandarin"...
260 ; GLOBALS : none
261 ; CALL : none
262 ; USER : printe
263
264 (defun imprimer-anglais (etape2)
265   (mapcar #'caaar etape2) )
```

La fonction imprimer-cat

La fonction `imprimer-cat` isole et retourne uniquement les catégories grammaticales des mots anglais à partir de l'étape2.

La fonction utilise `mapcar` et `caadar` pour récupérer la catégorie grammaticale de chaque entrée anglaise.

```
267 ; NAME : imprimer-cat ; renvoie que les cat (on utilise etape2 car
268 ; à partir de etape3 on a modifié la construction du dico > en vue des comparaisons)
269 ; ARGS : liste
270 ; USAGES : (imprimer-cat etape2) (imprimer-cat etape2) ("{adv}" "{proverb}" "{prefix}" ...
271 ; GLOBALS : none
272 ; CALL :
273 ; USER : printe
274
275 (defun imprimer-cat (etape2)
276   (mapcar #'caadar etape2) )
```

La fonction imprimer-espagnol

La fonction `imprimer-espagnol` récupère et renvoie les traductions espagnoles à partir d'`étape2`. La fonction parcourt la liste `étape2` et extrait la deuxième partie de chaque élément (les traductions espagnoles) pour les ajouter à une nouvelle liste.

```
278 ; NAME : imprimer-espagnol ; renvoie que les mots en esp
279 ;      (on utilise etape2 car à partir de etape3
280 ;      on a modifié la construction du dico > en vue des comparaisons de .txt)
281 ; ARGS : liste &aux resultat
282 ; USAGES : (imprimer-espagnol etape2) > ; (("hacia atrás") ("un día más un día menos")
283 ;      ("hiper-" ("ico") ("majestad {f}") ("mandarín {m}", chino mandarín {m})) ("bah")
284 ; GLOBALS : none
285 ; CALL :
286 ; USER : printe
287
288 (defun imprimer-espagnol (etape2 &aux resultat)
289   (loop for element in etape2 do ; pour chaque élément de etape2
290     (push (nth 1 element) resultat) ) ; on ajoute le cdr à resultat
291   (reverse resultat) ) ; on renvoie la liste inversée
```

La fonction printe

Les trois fonctions (`imprimer-anglais`, `imprimer-cat` et `imprimer-espagnol`) sont utilisées par la fonction `printe` pour reconstruire le dictionnaire.

La fonction `printe` reconstruit la liste originale de mots à partir de `etape2` (avant les modifications structurales que nous avons faites pour permettre l'ajout des catégories grammaticales manquantes pour les traductions).

La fonction `printe` collecte les mots anglais, leurs catégories et leurs traductions espagnoles en utilisant les fonctions `imprimer-anglais`, `imprimer-cat` et `imprimer-espagnol`, puis les combine en une liste.

```
295 ; NAME : printe ; reconstruit la liste de départ à partir de étape 2
296 ;      (car au delà on a modifié la construction du dico
297 ;      pour supprimer les virgules entre les trads)
298 ; ARGS : liste &aux parties-anglaises parties-cat parties-espagnoles result
299 ; USAGES : (printe etape2)> ("aback" "{adv}" "hacia atrás")
300 ; GLOBALS : none
301 ; CALL : imprimer-anglais, imprimer-cat, imprimer-espagnol
302 ; USER : main
303
304 (defun printe (liste &aux parties-anglaises parties-cat parties-espagnoles result)
305   (setq parties-anglaises (imprimer-anglais liste)) ; reconstruction des parties anglaises
306   (setq parties-cat (imprimer-cat liste)) ; reconstruction des catégories
307   (setq parties-espagnoles (imprimer-espagnol liste)) ; reconstruction des parties espagnoles
308   (setq result nil) ; liste vide pour les résultats
309   (loop while parties-anglaises do
310     (push (list (car parties-anglaises) (car parties-cat) (caar parties-espagnoles)) result)
311     ; on ajoute le mot anglais + sa cat + sa trad esp à résultat
312     (setq parties-anglaises (cdr parties-anglaises))
313     (setq parties-cat (cdr parties-cat))
314     (setq parties-espagnoles (cdr parties-espagnoles)) )
315   (reverse result) ) ; on inverse résultat
```

Afin d'illustrer nos fonctions, nous définissons la variable `myprint` qui correspond à l'appel de la fonction `printe` avec la liste `etape2` en argument. Cette variable globale est uniquement illustrative et n'est pas néces-

saire au bon fonctionnement du programme.

```
320 ; NAME : myprint ; la reconstruction du dico à partir de étape2
321 ; USAGES : (("aback" "{adv}" "hacia atrás")...
322 ; USER :
323 ; (setq myprint (printe etape2))
```

```
[6]> (setq myprint (printe etape2))
(("a" "{art}" "uno {art}, una {art}") ("Aachen" "{prop}" "Aquisgrán")
 ("aah" "{interj}" "¡ah!, ¡ay!") ("Aalenian" "{adj}" "aaleniano")
 ("Aalenian" "{prop}" "aaleniano {m}") ("aalii" "{n}" "chirca {f}")
 ("aardvark" "{n}" "cerdo hormiguero {m}")
 ("aardwolf" "{n}" "proteles {m}, lobo de tierra {m}")
 ("Aaron" "{prop}" "Aarón") ("abaca" "{n}" "abacá {m}")
 ("abacist" "{n}" "abacista {m}{f}") ("aback" "{adv}" "hacia atrás")
 ("abacus" "{n}" "ábaco {m}") ("Abadan" "{prop}" "Abadán")
 ("abaft" "{adv}" "a popa") ("abaft" "{prep}" "a popa")
 ("abaka" "{n}" "abacá {m}"))
```

FIGURE 7 – La variable myprint correspond à l'appel de la fonction printe en lui passant etape2 en argument.

La fonction imprimer

La fonction `imprimer`, est qui appelée par la fonction `main` écrit la liste reconstruite dans le fichier de sortie. `imprimer` écrit la liste formatée par `mon_format`.

```
326 ; NAME : imprimer ; imprimer la liste reconstruite dans le fichier
327 ; ARGS : liste (étape2) + fichier sortie
328 ; USAGES : (imprimer myprint "test.txt") > nil
329 ; GLOBALS : none
330 ; CALL : mon_format
331 ; USER : main
332
333 (defun imprimer (liste fichier)
334   (with-open-file (out fichier
335     :direction :output
336     :if-exists :supersede )
337     (mon_format liste out) ) )
```

La fonction mon_format

La fonction `mon_format` formate et écrit le contenu de notre liste dans un fichier.

La fonction parcourt la liste et pour chaque élément, écrit le mot anglais, sa catégorie et sa traduction espagnole sur une nouvelle ligne dans le fichier de sortie.

```
339 ; NAME : mon_format ; formatage de la liste pour l'impression
340 ; ARGS : liste (étape2) + fichier sortie
341 ; USAGES : (mon_format myprint out) > nil
342 ; GLOBALS : none
343 ; CALL : none
344 ; USER : imprimer
345
```

```

346 (defun mon_format (liste fichier_sortie)
347   (if (atom liste) ; si fin de la liste
348       nil ; on renvoie nil
349       (loop for item in liste do ; sinon pour chaque elt de la L
350           (format fichier_sortie "~a ~a ~a~%"
351               (car item) ; mot anglais
352               (cadr item) ; cat
353               (caddr item) ) ) ) ; partie espagnole

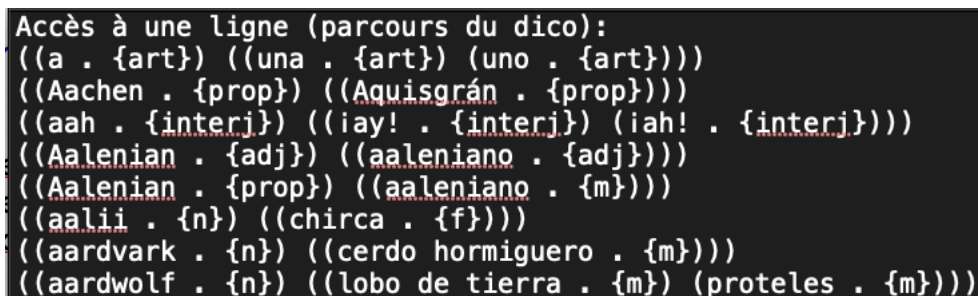
```

1.1.7 L'accès aux données

Dans le cadre des tests que nous effectuons, nous allons accéder aux différentes données. L'ensemble des sous-fonctions présentées dans un premier temps seront réutilisées dans la fonction centralisatrice `test`.

La fonction acces-ligne

La fonction `acces-ligne` parcourt et imprime chaque ligne de `etape4` dans un fichier. Cette fonction sert à parcourir le dictionnaire finalisé et à imprimer chaque ligne (c'est-à-dire le mot en anglais et sa catégorie grammaticale ainsi que ses traductions en espagnol).



```

Accès à une ligne (parcours du dico):
(a . {art}) ((una . {art}) (uno . {art}))
(Aachen . {prop}) ((Aquisgrán . {prop}))
(aah . {interj}) ((iay! . {interj}) (iah! . {interj}))
(Aalenian . {adj}) ((aaleniano . {adj}))
(Aalenian . {prop}) ((aaleniano . {m}))
(aalii . {n}) ((chirca . {f}))
(aardvark . {n}) ((cerdo hormiguero . {m}))
(aardwolf . {n}) ((lobo de tierra . {m}) (proteles . {m}))

```

FIGURE 8 – Le fichier ligne.txt qui résulte de l'utilisation de la fonction `acces-ligne` avec la liste `etape4` en argument par la fonction `test`

```

356 ; NAME : acces-ligne ; parcours le dictionnaire
357 ; ARGS : etape4 stream
358 ; USAGES : (acces-ligne etape4 acces-ligne.txt) > Accès à une ligne (parcours du dico):
359 ;      ((a . {art}) ((una . {art}) (uno . {art})))
360 ; GLOBALS : none
361 ; CALL : none
362 ; USER : test
363
364 (defun acces-ligne (etape4 stream)
365   (format stream "Accès à une ligne (parcours du dico):~%"
366       (loop for ligne in etape4 do
367           (format stream "~a~%" ligne) ) )

```

La fonction acces-anglais

La fonction `acces-anglais` isole et imprime uniquement la partie anglaise de chaque entrée de `etape4`. La partie anglaise de chaque entrée correspond au `car` de chaque ligne.

```
Accès à l'anglais (car de la ligne):
(a . {art})
(Aachen . {prop})
(aah . {interj})
(Aalenian . {adj})
(Aalenian . {prop})
(aalii . {n})
(aardvark . {n})
(aardwolf . {n})
(Aaron . {prop})
(abaca . {n})
```

FIGURE 9 – Le fichier anglais.txt qui résulte de l'utilisation de la fonction acces-anglais avec la liste etape4 en argument par la fonction test

```
369 ; NAME : acces-anglais (car de la ligne)
370 ; ARGS : etape4 stream
371 ; USAGES : (acces-anglais etape4 acces-anglais.txt) ; Accès à l'anglais (car de la ligne):
372 ;      (a . {art})
373 ; GLOBALS : none
374 ; CALL : none
375 ; USER : test
376
377 (defun acces-anglais (etape4 stream)
378   (format stream "Accès à l'anglais (car de la ligne):~%"
379     (loop for ligne in etape4 do
380       (format stream "~a~%" (car ligne)) ) )
```

La fonction acces-espagnol

La fonction `acces-espagnol` extrait et imprime la partie espagnole de chaque entrée d'`etape4`. Cette fonction imprime le `cadr` (c'est-à-dire le second élément) de chaque ligne d'`etape4`. Cela représente les traductions espagnoles du mot anglais.

```
Accès à l'espagnol (cadr de la ligne):
((una . {art}) (uno . {art}))
((Aquisgrán . {prop}))
((¡ay! . {interj}) (¡ah! . {interj}))
((aaleniano . {adj}))
((aaleniano . {m}))
((chirca . {f}))
((cerdo hormiguero . {m}))
((lobo de tierra . {m}) (proteles . {m}))
```

FIGURE 10 – Le fichier espagnol.txt qui résulte de l'utilisation de la fonction acces-espagnol avec la liste etape4 en argument par la fonction test

```
382 ; NAME : acces-espagnol (cadr de la ligne)
383 ; ARGS : etape4 stream
384 ; USAGES : (acces-espagnol etape4 stream) > Accès à l'espagnol (cadr de la ligne):
385 ;      ((una . {art}) (uno . {art}))
386 ; GLOBALS : none
387 ; CALL : none
388 ; USER : test
389
390 (defun acces-espagnol (etape4 stream)
391   (format stream "Accès à l'espagnol (cadr de la ligne):~%"
```

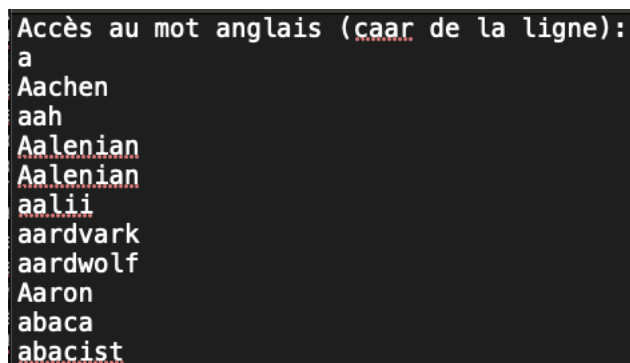
```

392 (loop for ligne in etape4 do
393   (format stream "~a~%" (cadr ligne)) ) )

```

La fonction acces-mot-anglais

La fonction `acces-mot-anglais` imprime le mot anglais de chaque entrée. Cela correspond au premier élément de chaque première paire.



```

Accès au mot anglais (caar de la ligne):
a
Aachen
aah
Aalenian
Aalenian
aalii
aardvark
aardwolf
Aaron
abaca
abacist

```

FIGURE 11 – Le fichier mot-anglais.txt qui résulte de l'utilisation de la fonction `acces-mot-anglais` avec la liste `etape4` en argument par la fonction `test`

```

395 ; NAME : acces-mot-anglais (caar de la ligne)
396 ; ARGS : etape4 stream
397 ; USAGES : (acces-mot-anglais etape4 stream) > Accès au mot anglais (caar de la ligne): a
398 ; GLOBALS : none
399 ; CALL : none
400 ; USER : test
401
402 (defun acces-mot-anglais (etape4 stream)
403   (format stream "Accès au mot anglais (caar de la ligne):~%"
404     (loop for ligne in etape4 do
405       (format stream "~a~%" (caar ligne)) ) )

```

La fonction acces-cat-anglais

La fonction `acces-cat-anglais` imprime la catégorie grammaticale du mot anglais de chaque entrée. Cela correspond au deuxième élément de la première paire.

```

407 ; NAME : acces-cat-anglais (cdar de la ligne)
408 ; ARGS : etape4 stream
409 ; USAGES : (acces-cat-anglais etape4 stream) > Accès à la catégorie de l'anglais
410 ;   (cdar de la ligne): {art}
411 ; GLOBALS : none
412 ; CALL : none
413 ; USER : test
414
415 (defun acces-cat-anglais (etape4 stream)
416   (format stream "Accès à la catégorie de l'anglais (cdar de la ligne):~%"
417     (loop for ligne in etape4 do
418       (format stream "~a~%" (cdar ligne)) ) )

```

```
Accès à la catégorie de l'anglais (cadr de la ligne):
{art}
{prop}
{interj}
{adj}
{prop}
{n}
{n}
{n}
{prop}
{n}
{n}
{adv}
{n}
```

FIGURE 12 – Le fichier cat-anglais.txt qui résulte de l'utilisation de la fonction acces-cat-anglais avec la liste etape4 en argument par la fonction test

La fonction acces-traduction

La fonction `acces-traduction` parcourt et imprime chaque traduction espagnole et sa catégorie (chaque sous-paire du deuxième élément de la ligne).

La fonction boucle à travers les entrées de `etape4` et pour chaque entrée boucle à travers les traductions espagnoles en imprimant chaque sous-paire.

```
Accès à une traduction (parcours du cadr de la ligne):
(una . {art})
(uno . {art})
(Aquisgrán . {prop})
(iay! . {interj})
(iah! . {interj})
(aaleniano . {adj})
(aaleniano . {m})
(chirca . {f})
(cerdo hormiguero . {m})
(lobo de tierra . {m})
(proteles . {m})
```

FIGURE 13 – Le fichier traduction.txt qui résulte de l'utilisation de la fonction acces-traduction avec la liste etape4 en argument par la fonction test

```
421 ; NAME : acces-traduction (parcours du cadr de la ligne)
422 ; ARGS : etape4 stream
423 ; USAGES : (acces-traduction etape4 stream) > Accès à une traduction
424 ; (parcours du cadr de la ligne): (una . {art})
425 ; GLOBALS : none
426 ; CALL : none
427 ; USER : test
428
429 (defun acces-traduction (etape4 stream)
430 (format stream "Accès à une traduction (parcours du cadr de la ligne):~%"
431 (loop for ligne in etape4 do
432 (loop for traduction in (cadr ligne) do
433 (format stream "~a~%" traduction) ) ) )
```

La fonction acces-mot-espagnol

La fonction `acces-mot-espagnol` imprime seulement le mot espagnol (le premier élément de chaque sous-paire) pour chaque traduction.

```
Accès à un mot espagnol (car de la sous-liste):
una
uno
Aquisgrán
¡ay!
¡ah!
aaleniano
aaleniano
chirca
cerdo hormiguero
lobo de tierra
proteles
Aarón
```

FIGURE 14 – Le fichier mot-espagnol.txt qui résulte de l'utilisation de la fonction acces-mot-espagnol avec la liste etape4 en argument par la fonction test

```
435 ; NAME : acces-mot-espagnol (parcours car de la sous-liste)
436 ; ARGS : etape4 stream
437 ; USAGES : (acces-mot-espagnol etape4 stream) > Accès à un mot espagnol
438 ;          (car de la sous-liste): una
439 ; GLOBALS : none
440 ; CALL : none
441 ; USER : test
442
443 (defun acces-mot-espagnol (etape4 stream)
444   (format stream "Accès à un mot espagnol (car de la sous-liste):~%" )
445   (loop for ligne in etape4 do
446     (loop for traduction in (cadr ligne) do
447       (format stream "~a~%" (car traduction)) ) ) )
```

La fonction acces-cat-trad

La fonction `acces-cat-trad` imprime la catégorie grammaticale de chaque traduction espagnole (le deuxième élément de chaque sous-paire) pour chaque traduction.

```
Accès à la catégorie d'une traduction (cdr de la sous-liste):
{art}
{art}
{prop}
{interj}
{interj}
{adj}
{m}
{f}
{m}
{m}
{m}
{prop}
{m}
{m}
{adv}
```

FIGURE 15 – Le fichier cat-trad.txt qui résulte de l'utilisation de la fonction acces-cat-trad avec la liste etape4 en argument par la fonction test

```
449 ; NAME : acces-cat-trad (parcours cdr de la sous-liste)
450 ; ARGS : etape4
451 ; USAGES : (acces-cat-trad etape4 stream) > Accès à la catégorie d'une traduction
452 ;          (cdr de la sous-liste): {art}
453 ; GLOBALS : none
454 ; CALL : none
```

```

455 ; USER : test
456
457 (defun acces-cat-trad (etape4 stream)
458   (format stream "Accès à la catégorie d'une traduction (cdr de la sous-liste):~%" )
459   (loop for ligne in etape4 do
460     (loop for traduction in (cadr ligne) do
461       (format stream "~a~%" (cdr traduction)) ) ) )

```

La fonction test

La fonction `test` est appelée au sein de la fonction `main`. Elle permet de vérifier les données obtenues. Chaque fichier créé va contenir une partie spécifique d'`etape4`.

L'accès à la ligne complète : le fichier `ligne.txt` est créé ou écrasé s'il existe déjà. La fonction `acces-ligne` est appelée.

L'accès à la partie anglaise : le fichier `anglais.txt` est créé ou écrasé s'il existe déjà. La fonction `acces-anglais` est appelée.

L'accès à la partie espagnole : le fichier `espagnol.txt` est créé ou écrasé s'il existe déjà. La fonction `acces-espagnol` est appelée.

L'accès au mot anglais : le fichier `mot-anglais.txt` est créé ou écrasé s'il existe déjà. La fonction `acces-mot-anglais` est appelée.

L'accès à la catégorie du mot anglais : le fichier `cat-anglais.txt` est créé ou écrasé s'il existe déjà. La fonction `acces-cat-anglais` est appelée.

L'accès aux traductions espagnoles : le fichier `traduction.txt` est créé ou écrasé s'il existe déjà. La fonction `acces-traduction` est appelée.

L'accès aux mots espagnols : le fichier `mot-espagnol.txt` est créé ou écrasé s'il existe déjà. La fonction `acces-mot-espagnol` est appelée.

L'accès aux catégories grammaticales des mots espagnols : le fichier `cat-trad.txt` est créé ou écrasé s'il existe déjà. La fonction `acces-cat-trad` est appelée.

```

464 ; NAME : test ; pour vérifier les accès aux données
465 ;      imprime chaque fonction dans un fichier
466 ; ARGS : etape4
467 ; USAGES : (test etape4) > nil
468 ; GLOBALS : none
469 ; CALL : acces-ligne, acces-anglais, acces-espagnol, acces-mot-anglais
470 ; acces-cat-anglais, acces-traduction, acces-mot-espagnol, acces-cat-trad
471 ; USER : main
472
473 (defun test (etape4)
474   (with-open-file
475     (stream "ligne.txt"
476       :direction :output
477       :if-exists :supersede
478       :if-does-not-exist :create )
479     (acces-ligne etape4 stream) )
480   (with-open-file
481     (stream "anglais.txt"
482       :direction :output
483       :if-exists :supersede
484       :if-does-not-exist :create )

```

```

485 (acces-anglais etape4 stream) )
486 (with-open-file
487   (stream "espagnol.txt"
488     :direction :output
489     :if-exists :supersede
490     :if-does-not-exist :create )
491   (acces-espagnol etape4 stream) )
492 (with-open-file
493   (stream "mot-anglais.txt"
494     :direction :output
495     :if-exists :supersede
496     :if-does-not-exist :create )
497   (acces-mot-anglais etape4 stream) )
498 (with-open-file
499   (stream "cat-anglais.txt"
500     :direction :output
501     :if-exists :supersede
502     :if-does-not-exist :create )
503   (acces-cat-anglais etape4 stream) )
504 (with-open-file
505   (stream "traduction.txt"
506     :direction :output
507     :if-exists :supersede
508     :if-does-not-exist :create )
509   (acces-traduction etape4 stream) )
510 (with-open-file
511   (stream "mot-espagnol.txt"
512     :direction :output
513     :if-exists :supersede
514     :if-does-not-exist :create )
515   (acces-mot-espagnol etape4 stream) )
516 (with-open-file
517   (stream "cat-trad.txt"
518     :direction :output
519     :if-exists :supersede
520     :if-does-not-exist :create )
521   (acces-cat-trad etape4 stream) ) )

```

1.1.8 Le programme de traduction

Ce prototype Lisp présente un programme de traduction de l'anglais vers l'espagnol. Nous avons donc créé une fonction `traduction` qui prend en argument un mot et sa catégorie (entre `{}`). La fonction retourne la ou les traductions ainsi que leurs catégories si elles existent. Sinon, la fonction renvoie un message précisant qu'aucune traduction n'est disponible.

La fonction `format` est utilisée afin de présenter les résultats comme nous le souhaitons. La seule solution que nous avons trouvée afin de ne pas avoir de nil suite à l'affichage a été d'utiliser (`values`) qui renvoie une liste vide.

La fonction `traduction` est appelée au toplevel. Elle est destinée à rechercher et afficher les traductions espagnoles d'un mot anglais selon sa catégorie grammaticale.

La fonction reconstruit `etape1`, `etape2`, `etape3` et `etape4` que nous avons présentées précédemment.

`etape4` correspond à notre dictionnaire finalisé avec une catégorie grammaticale pour les différentes traductions.

La fonction boucle sur `etape4` et pour chaque ligne vérifie si le mot anglais (le `caar` de la ligne) et sa catégorie (le `cdar` de la ligne) correspondent aux arguments de la fonction (c'est-à-dire le mot anglais recherché et sa catégorie grammaticale).

Si une correspondance est trouvée, on passe `mot-trouvé` à `t`. La fonction entre alors dans une autre boucle pour chaque traduction espagnole (c'est-à-dire le `cadr` de la ligne). On imprime alors chaque traduction avec sa catégorie.

Si aucune correspondance n'est trouvée, un message indiquant qu'aucune traduction n'est disponible pour le mot et sa catégorie apparaît.

Parmi les différentes catégories grammaticales pouvant être utilisées, les plus fréquentes sont : `{n}`, `{v}`, `{adj}`, `{adv}`.

```
Break 1 [8]> (traduction 'love '{n})
amor {m}
cariño {m}

Break 1 [8]> (traduction 'love '{v})
hacer el amor {v}
encantar {v}
amar {v}
querer {v}
gustar {v}
adorar {v}

Break 1 [8]> (traduction 'loove '{v})
Il n'y a pas de traduction disponible pour le mot loove de catégorie {v}
```

FIGURE 16 – Utilisation de la fonction `traduction`

```
553 ; NAME : traduction
554 ; ARGS : mot categorie &aux etape1 etape2 etape3 etape4 mot-trouve
555 ; USAGES : (traduction 'a '{art}) > una {art} uno {art}
556 ; GLOBALS : none
557 ; CALL : none
558 ; USER : toplevel
559
560 (defun traduction (mot categorie &aux etape1 etape2 etape3 etape4 mot-trouve)
561   (setq etape1 (lire-fichier "mon_dico.txt")) ; lit le fichier ligne par ligne
562   (setq etape2 (parentheses etape1)) ; ajoute les parenthèses
563   (setq etape3 (structure_en_cours etape2)) ; isole chaque différente trad
564   (setq etape4 (forme_finale etape3 etape3)) ; ajoute les cat aux différentes trad
565   (loop for ligne in etape4 do ; pour chaque ligne du dico
566     (when (and (string-equal (caar ligne) mot) (string-equal (cadr ligne) categorie))
567       ; si le mot et la catégorie correspondent
568       (setq mot-trouve t) ; on a trouvé le mot
569       (loop for traduction in (cadr ligne) do ; pour chaque traduction
570         (format t "~A ~A~%" (car traduction) (cdr traduction)) ) )
571       ; on imprime la traduction
572       (unless mot-trouve ; si le mot n'a pas été trouvé
573         (format t "Il n'y a pas de traduction disponible pour le mot ~A de catégorie ~A~%" mot categorie) )
574       (values) ) ; on renvoie rien (pour éviter le nil)
```

1.2 Programme en C - Analyse des données

Nous allons utiliser pour notre programme de traduction un dictionnaire anglais-espagnol que nous avons trouvé en ligne (source : <https://github.com/mananoreboton/en-es-en-Dic/blob/master/src/main/resources/dic/en-es.xml>).

Lorsque nous avons dû choisir notre dictionnaire, le site proposé par le cours (polyglotte.tuxfamily.org) était inaccessible. C'est la raison pour laquelle nous avons décidé d'utiliser le dictionnaire mentionné précédemment, bien qu'il n'ait pas été idéal. Il contenait initialement 45 592 entrées mais un grand nombre d'entre elles étaient inexploitable (doublons, mot sans traduction, etc). Après les traitements successifs de notre dictionnaire, nous avons conservé 30 664 entrées. Il demeure probablement certaines corrections à apporter au dictionnaire car nous n'avons pas été en mesure de corriger chaque entrée individuellement.

Les entrées de notre dictionnaire se présentent de la façon suivante :

mot anglais {cat_obligatoire} trad1 {cat_optionnelle}, trad_suivante {cat_optionnelle}

Chaque mot anglais est associé à une catégorie qui se trouve entre { }. Notre dictionnaire présente 31 catégories différentes (comme nous le détaillerons plus tard). Un mot anglais peut avoir une ou plusieurs traductions en espagnol. Les traductions sont séparées par des virgules. Les mots en espagnol peuvent présenter, ou non, une catégorie. Lorsqu'aucune catégorie n'est définie, la catégorie du mot anglais est associée à la traduction.

```
ahead {adv} enfrente de
aid {n} ayuda {f}, asistente, ayudante, remedio {m}, auxilio {m}, recurso {m}
aid {v} ayudar
Aida {prop} Aída
aide {n} ayudante
aide-de-camp {n} edecan {m}
AIDS {acronym} SIDA {m}, sida {m}
aiglet {n} herrete {m}
ail {n} enfermedad {f}
ail {v} molestar, causar dolor a, adolecer
aileron {n} alerón {m}
ailment {n} enfermedad {f}, dolencia {f}, achaque {m}
aim {n} intención {m}, objetivo {m}
aim {v} lazard, apuntar, dirigir
aimless {adj} sin rumbo, sin objeto
```

FIGURE 17 – La présentation de notre dictionnaire anglais-espagnol.

Nous avons défini dans un fichier `structure.h` les structures de notre dictionnaire.

La `struct Doublet` (et `list` un pointeur sur un `Doublet`) est la structure de base pour les listes. Elle contient un pointeur `car` vers n'importe quel type de données et un pointeur `cdr` vers un autre `doublet`.

La `struct MotDictionnaire` (et `une_entree` un pointeur sur `MotDictionnaire`) représente une entrée dans le dictionnaire. Elle contient un pointeur `anglais` vers une structure `base` qui stocke le mot anglais et sa catégorie. Elle contient un pointeur `traduction` vers une structure `Trad` qui gère les traductions en espagnol du mot anglais. Enfin, il y a un pointeur `cdr` qui permet de chaîner vers l'entrée suivante du dictionnaire.

La `struct Base` (et `forme_anglais` et `forme_espagnol` qui sont des pointeurs sur `Base`) stocke les détails d'un mot, qu'il soit en anglais ou en espagnol. La structure contient des champs `mot` et `original` pour stocker le mot et sa forme originale (pour permettre de reconstruire le fichier lors des test). Enfin, il y a un pointeur vers la structure `Category` qui définit la catégorie grammaticale du mot.

La `struct Trad` (et `trad_espagnol` et `trad_anglais` qui sont des pointeurs sur `Trad`) gère la traduction d'un mot. Néanmoins, seule la traduction espagnole est actuellement traitée. L'inversion du dictionnaire (permettant également la traduction de l'espagnol vers l'anglais) n'a pas été implémentée. La structure `Trad` contient un pointeur vers une structure `Base` pour stocker les détails du mot. En outre, il y a un pointeur `cdr` permettant de chaîner vers une autre traduction, lorsqu'un même mot présente plusieurs traductions.

Finalement, la `struct Category` (et `la_categorie` un pointeur sur `Category`) définit la catégorie grammaticale du mot. La structure contient un champ `chaîne` pour le nom de la catégorie et un champ `cat` pour l'entier qui définit la catégorie.

```
17 typedef struct Doublet {
18     void * car ;
19     struct Doublet * cdr ;
20 } * list ; // structure de Base ; list est un pointeur sur un doublet
21
22 typedef struct MotDictionnaire {
23     struct Base * anglais;
24     struct Trad * traduction;
25     struct MotDictionnaire * cdr ;
26 } * une_entree ;
27
28 typedef struct Base {
29     char* mot;
30     char* original;
31     struct Category * categorie;
32 } * forme_anglais, * forme_espagnol ;
33
34 typedef struct Trad {
35     struct Base * espagnol;
36     struct Trad * cdr;
37 } * trad_espagnol, * trad_anglais ;
38
39 typedef struct Category{
40     char* chaîne;
41     int cat;
42 } * la_categorie ;
```

1.3 Programme en C - Présentation du programme

Le programme de notre traducteur en ligne a été divisé en plusieurs fichiers :

- Le fichier `sys.h` est un fichier que l'on modifie au fur et à mesure de nos projets et qui contient les bibliothèques dont nous avons eu besoin lors de nos programmes en C.
- Le fichier `structure.h` qui contient les définitions des structures créées (présentées précédemment) ainsi que les prototypes des fonctions.
- Le fichier `dictionnaire.c` est le fichier qui contient la fonction `main`. C'est là qu'on trouve le coeur de notre programme.
- Le fichier `category.c` gère l'attribution d'un identifiant numérique pour chaque type de catégorie.
- Le fichier `test.c` gère la création des fichiers de test et le lancement des tests. Pour être utilisé, le programme doit être compilé avec l'instruction de compilation `-DTEST`.

Nous allons présenter les fichiers les uns après les autres. Bien entendu, nous détaillerons les différentes fonctions du fichier `dictionnaire.c`.

Le fichier sys.h

Le fichier `sys.h` contient les différentes bibliothèques dont nous nous servons régulièrement pour nos programmes. Nous rajoutons les bibliothèques manquantes au fur et à mesure de nos nouveaux projets de programmation.

Voici l'intégralité du fichier :

```
1 #include <assert.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <unistd.h>
6 #include <sys/stat.h>
7 #include <sys/types.h>
8 #include <fcntl.h>
9 #include <sys/wait.h>
10 #include <stdbool.h>
11 #include <errno.h>
12 #include <time.h>
13 #include <signal.h>
14 # include <ctype.h>
```

Le fichier structure.h

Le fichier `structure.h` définit les structures de données et les prototypes de fonction nécessaires à la création de notre dictionnaire.

Le fichier définit le type `string` comme alias pour `char *`.

Des macros permettant d'accéder aux éléments `car` et `cdr` d'une structure `Doublet` sont également définies.

```
1 /* # Nom ..... : structure.h
2 # Rôle ..... : proto des fonctions et déclaration des structures
3 # Auteur ..... : AvriLe Floro
4 # Version ..... : V0.1 du 17/11/23
5 # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 # Usage : fichier header */
7
8 typedef char * string ;
```

```

9 #define NULLCHAR "\\0"
10
11 #define car(doulet) ((doulet)->car)
12 // macro plutôt que fonction, pour écrire car(x) = ...
13 #define cdr(doulet) ((doulet)->cdr)
14 // macro plutôt que fonction, pour écrire cdr(x) = ...

```

Le fichier `structure.h` définit également les différentes structures de données utilisées dans la construction de notre dictionnaire. Nous les avons expliquées en détails lors de la partie **analyse de données**.

```

17 typedef struct Doublet {
18     void * car ;
19     struct Doublet * cdr ;
20 } * list ; // structure de Base ; list est un pointeur sur un doublet
21
22 typedef struct MotDictionnaire {
23     struct Base * anglais;
24     struct Trad * traduction;
25     struct MotDictionnaire * cdr ;
26 } * une_entree ;
27
28 typedef struct Base {
29     char* mot;
30     char* original;
31     struct Category * categorie;
32 } * forme_anglais, * forme_espagnol ;
33
34 typedef struct Trad {
35     struct Base * espagnol;
36     struct Trad * cdr;
37 } * trad_espagnol, * trad_anglais ;
38
39 typedef struct Category{
40     char* chaine;
41     int cat;
42 } * la_categorie ;

```

Finalement, le fichier `structure.h` contient les prototypes des fonctions utilisées dans notre programme.

```

47 list cons(void *, const list) ; /* construction liste */
48
49 list reverse(list L) ; /* inversion liste */
50
51 une_entree div_eng_esp(list L); /* renvoie l'entrée */
52
53 forme_anglais eng_cat(string anglais); /* renvoie la forme anglaise */
54
55 trad_espagnol spanish(string esp, string cat); /* renvoie la trad esp */
56
57 forme_espagnol subdiv_esp(string token, string cat); /* renvoie la forme esp */
58
59 la_categorie assoc_categorie(string cat_originale); /* association catégorie */
60
61 void initialisation_fichiers(); /* initialisation préalable des fichiers en W */
62
63 void creation_tests(une_entree dictionnaire); /* création des tests si -DTEST */
64
65 void traduction(une_entree dictionnaire); /* programme de la traduction */

```

Le fichier dictionnaire.c

Le fichier `dictionnaire.c` est le fichier principal de notre programme, c'est le fichier qui contient la fonction `main`.

Notre programme a été organisé en tiroir. C'est-à-dire que chacune des fonctions de traitement des données sera chargée du traitement d'un niveau de données en particulier et fera appel à une fonction pour traiter le niveau de données suivant.

Le dictionnaire entièrement traité est renvoyé par la fonction `div_eng_esp`. Cette fonction n'est en charge que du remplissage des différents champs de `une_entree`, mais cette structure étant la plus globale, elle correspond à la construction complète du dictionnaire. Les structures subsidiaires sont complétées par des sous-fonctions.

La fonction main

La fonction `main` est le coeur de notre programme.

Elle charge en mémoire le dictionnaire au sein d'une liste. Chaque ligne du dictionnaire correspondra à un `car`. Pour ce faire, elle appelle la fonction `cons`. Pour conserver en mémoire le dictionnaire dans l'ordre alphabétique, elle inverse la liste ainsi obtenue grâce à la fonction `reverse`.

Finalement, est obtenu un dictionnaire sous forme d'entrées (`une_entree` est un pointeur vers `MotDictionnaire`) grâce à l'appel de la fonction `div_eng_esp`.

En outre, la fonction `main` se charge d'initialiser les fichiers de tests et de procéder aux tests lorsque les directives de compilation contiennent `-DTEST`. Si la directive de compilation `-DTEST` n'a pas été utilisée, alors notre programme entre dans une boucle d'appel de la fonction `traduction`, ce qui nous permet d'avoir un dictionnaire bilingue en ligne de commande.

```
1 /* # Nom ..... : dictionnaire.c
2 # Rôle ..... : un dictionnaire bilingue eng-esp
3 # Auteur ..... : AvriLe Floro
4 # Version ..... : V0.1 du 17/11/23
5 # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 # Usage : pour le dico bilingue > make > puis ./dictionnaire
7 # Usage : pour les tests de reconstruction > make -f makefile_tests
8 #      puis > ./dictionnaire_test */
9
10
11 #include "sys.h"
12 #include "structure.h"
13
14
15 int main(int argc, char *argv[]) {
16
17     # ifdef TEST /* si -DTEST on initialise les fichiers */
18     initialisation_fichiers();
19     # endif
20
21     FILE* f_entree = NULL;
22     char p_chaine[1000];
23
24     /* ouverture du fichier en lecture */
25     f_entree = fopen("mon_dico.txt", "r");
26
27     list L = NULL ;
28
29     if (f_entree != NULL) {
30         /* mise en mémoire du dico */
31         while (fgets(p_chaine, sizeof(p_chaine), f_entree)) {
32             p_chaine[strcspn(p_chaine, "\n")] = '\0'; /* on supprime le \n */
33             L = cons(strdup(p_chaine), L); /* ajoute copie ligne à la liste */

```

```

34     }
35     fclose(f_entree);
36 }
37
38 /* inversion de la liste pour conserver l'ordre */
39 L = reverse(L);
40 /* obtention de notre dictionnaire traité */
41 une_entree dictionnaire = div_eng_esp(L);
42
43 #ifdef TEST /* si -DTEST on fait les tests */
44     creation_tests(dictionnaire);
45
46 #else /* sinon traducteur en ligne de commande */
47     while (true) {
48         traduction(dictionnaire);
49         printf("\n");
50     }
51 #endif
52     return 0;
53 }

```

La fonction traduction

La fonction `traduction` est la seconde fonction la plus importante de notre programme puisqu'elle est conçue pour rechercher la ou les traductions espagnoles d'un mot anglais d'une certaine catégorie au sein de notre dictionnaire.

La fonction reçoit en argument un `dictionnaire` de type `une_entree` (un pointeur vers `MotDictionnaire`) qui représente notre dictionnaire bilingue anglais-espagnol. L'obtention du dictionnaire grâce aux fonctions successives est détaillée ci-après.

La fonction demande à l'utilisateur d'entrer un mot anglais à traduire ainsi que sa catégorie grammaticale entre { }.

La ligne entière est récupérée par la fonction via `fgets` puis est divisée grâce à `strtok`. Le mot est traité pour supprimer l'espace final. La catégorie est traitée pour rajouter le { (supprimé lors de l'appel à `strtok`) et supprimer le dernier caractère.

La fonction vérifie l'entrée utilisateur et s'interrompt si l'utilisateur n'entre pas un mot et une catégorie entre { }.

La fonction parcourt le dictionnaire à la recherche d'une entrée qui correspond au mot et à la catégorie donnés.

Si une correspondance est trouvée, on affiche toutes les traductions espagnoles associées à cette entrée et on les sépare par des virgules.

Si aucune entrée n'est trouvée, un message indiquant qu'aucune traduction n'a été trouvée s'affiche.

```

55 /*; NAME : traduction ; donne l'esp d'un mot eng
56 ; ARGS : dictionnaire
57 ; USAGES : traduction(dictionnaire)
58 ; GLOBALS : none
59 ; CALL : none
60 ; USER : main */
61
62 void traduction(une_entree dictionnaire){
63
64     /* on récupère la ligne entière */
65     char ligne[100];
66     printf("Quel mot voulez-vous traduire ?\n");
67     fgets(ligne, sizeof(ligne), stdin);

```

```

68
69  /* traitement du mot */
70  string mot = strtok(ligne, "{"); /* séparation avant le { */
71  int len = strlen(mot);
72  mot[len-1]='\0'; /* on supprime l'espace */
73
74  /* traitement de la catégorie */
75  string cat = strtok(NULL, "");
76  if (cat == NULL){
77      printf("Usage: mot_anglais {catégorie_recherchée}\n");
78      return;
79  }
80
81  string categorie = malloc(strlen(cat) + 2); /*pour rajouter le { */
82  categorie[0]='\0'; /* initialisation */
83  strcat(categorie, "{"); /* rajout { */
84  strcat(categorie, cat); /* copie du strtok */
85  int len2 = strlen(categorie);
86  categorie[len2-1]='\0'; /* suppression dernier caractère*/
87
88
89  bool found = false;
90  une_entree p_dico = dictionnaire;
91  /* utilisation copie pour pas parcourir le dico original */
92
93  while (p_dico != NULL && found == false) { /* tant qu'il y a des mots */
94
95      if ((strcmp(p_dico->anglais->mot, mot) == 0) &&
96          (strcmp(p_dico->anglais->categorie->chaine, categorie) == 0)) {
97          /* si mot et catégorie sont identiques */
98
99
100         trad_espagnol temp = p_dico->traduction;
101         /* utilisation ptr temp pour conserver ordre original dico */
102
103         while (temp != NULL) { /* tant qu'il y a des trad du mot */
104             printf("%s %s", temp->espagnol->mot, temp->espagnol->categorie->chaine);
105             /* on les imprime */
106             if (temp->cdr != NULL) { /* et si ce n'est pas le dernier mot */
107                 printf(", "); /* on ajoute une virgule entre les trads */
108             }
109
110             temp = temp->cdr; /* on parcourt les trads */
111         }
112
113         found = true; /* arrêt du parcours du dico */
114         printf("\n");
115     }
116     p_dico = p_dico->cdr; /* on parcourt les mots du dico */
117 }
118
119 if (!found){
120     printf("Aucune traduction n'a été trouvée. \n");
121 }
122 }

```


La fonction cons

La fonction `cons` a été fournie par le cours. Elle est utilisée pour ajouter un nouvel élément au début d'une liste.

```
125 /*; NAME : cons ; ajout d'un élément au début d'une l
126 ; ARGS : elt, L
127 ; USAGES : cons(elt, L)
128 ; GLOBALS : none
129 ; CALL : none
130 ; USER : main */
131
132 list cons(void * elt, list L)
133 {
134     list Cons = malloc(sizeof(struct Doublet)) ;
135     Cons->car = elt;
136     Cons->cdr = L;
137
138     return Cons ;
139 }
```

La fonction reverse

La fonction `reverse` a été réutilisée des exercices précédent. Elle inverse l'ordre des éléments d'une liste qu'elle reçoit en argument. Elle modifie la liste d'origine.

```
141 /*; NAME : reverse ; inverse ordre liste
142 ; ARGS : L
143 ; USAGES : reverse(L)
144 ; GLOBALS : none
145 ; CALL : none
146 ; USER : main */
147
148 list reverse(list L) {
149     list R = NULL; // liste vide
150     while (L != NULL) {
151         list temp = cdr(L); // sauvegarde du cdr de L
152         cdr(L) = R; // le cdr de L pointe sur R
153         R = L; // R pointe sur L
154         L = temp; // L pointe sur le prochain élt de L
155     }
156     return R;
157 }
```

La fonction div_eng_esp

La fonction `div_eng_esp` traite une liste de mots et divise chaque entrée en deux parties : une partie correspond à l'anglais et la seconde partie correspond à la traduction en espagnol. La fonction prend en argument une liste et renvoie `debut_dico`. Cela correspond au dictionnaire construit et rempli sous forme de liste.

On initialise `debut_dico` à `NULL`. Il sera utilisé, ainsi que `precedent`, pour construire notre dictionnaire à partir de la liste.

La fonction boucle sur la liste qu'elle prend en argument. Pour chaque élément de la liste, on alloue de la mémoire pour une nouvelle instance de la structure `MotDictionnaire` et on utilise un pointeur `un_mot` (`une_entree` est un pointeur vers `MotDictionnaire`).

Chaque élément de la liste correspond à une ligne du fichier texte `mon_dico.txt`. Grâce à `strtok`, on divise la partie anglaise et la partie espagnole.

La partie anglaise est traitée pour rajouter le `}` qui a été supprimé par `strtok`. La chaîne `engl` ainsi formée est passée en argument à la fonction `eng_cat` qui construit l'entrée anglaise du mot.

La chaîne `esp` qui correspond à la partie espagnole de la ligne est passée à la fonction `spanish` qui construit l'entrée espagnole du mot.

Les différentes entrées du dictionnaire (`un_mot`) sont connectées entre elles par leur `cdr` (elles forment une liste) pour former le dictionnaire final.

```
160 /*; NAME : div_eng_esp ; traitement données complet dico
161 ; ARGS : L
162 ; USAGES : div_eng_esp(L)
163 ; GLOBALS : none
164 ; CALL : eng_cat, spanish
165 ; USER : main */
166
167 une_entree div_eng_esp(list L)
168 {
169     une_entree debut_dico = NULL; /* pour connecter les cdr */
170     une_entree precedent = NULL;
171
172     while (L != NULL){ /* parcourt de la liste */
173
174         une_entree un_mot = malloc(sizeof(struct MotDictionnaire));
175         string ligne = (string)(car(L)); /* mise en mémoire des lignes */
176
177         /* division des catégories */
178         /* anglais */
179         string eng = strtok(ligne, "}"); /* premier token jusqu'à \} */
180         /* espagnol */
181         string esp = strtok(NULL, ""); /* 2e token jusqu'à la fin */
182
183         /* traitement de l'anglais */
184         string engl = malloc(strlen(eng) + 2); /* pour rajouter le } */
185         strcpy(engl, eng);
186         strcat(engl, "}"); /* rajout { manquant */
187
188         /* formation entrée anglais */
189         un_mot->anglais = eng_cat(engl);
190
191         /* formation entrée traduction */
192         un_mot->traduction = spanish(esp, un_mot->anglais->categorie->chaîne);
193
194         /* connexion des entrées dico via le cdr */
195         if (debut_dico == NULL) { /* si début de la liste */
196             debut_dico = un_mot; /* 1er élément */
197         }
198         else { /* si pas début de la liste */
199             precedent->cdr = un_mot; /* ajout à la fin de la liste */
200         }
201         precedent = un_mot; /* parcours des élts */
202         L = cdr (L);
203     }
204 }
205
206 return debut_dico; /* renvoie le dico sous la bonne forme */
207 }
```

La fonction `eng_cat`

La fonction `eng_cat` traite une chaîne de caractères qui correspond à la partie anglaise d'une ligne de notre dictionnaire chargé en mémoire.

On alloue de la mémoire pour une nouvelle `forme_anglais` (qui est un pointeur vers `Base`).

Tout au long de la fonction, les chaînes de caractères sont dupliquées afin d'éviter la modification des chaînes originales.

La chaîne de caractères `english_part` est divisée en deux parties : la première partie correspond au mot anglais et la seconde partie correspond à la catégorie du mot. La division est réalisée en utilisant `strtok` à deux reprises.

La chaîne ainsi obtenue qui correspond au mot anglais est traitée afin de supprimer l'espace final conservé par `strtok`.

Un traitement est également appliqué à la chaîne correspondant à la catégorie du mot. On rajoute le `{` qui a été supprimé par `strtok`.

La fonction remplit les différents champs de `forme_anglais`. `forme_anglais` est composée d'un champ `mot`, qui a été isolé par `strtok`. Il y a un champ `original` qui permet de conserver la chaîne de caractères initiale avant traitement par `strtok`. Enfin, le champ `categorie` est rempli grâce au retour de la fonction `assoc_categorie` à laquelle on passe en argument la catégorie du mot anglais, isolée grâce au second appel de `strtok`.

La fonction renvoie `anglais` remplie.

Cette fonction a été la source d'un bug que nous avons eu des difficultés à identifier. Il s'agissait d'une erreur liée à la mauvaise allocation mémoire lors du travail sur les chaînes lorsque l'on rajoutait des caractères. Pour une raison inconnue, cette erreur ne s'est pas manifestée au début de notre travail sur le programme mais plus tard (à l'occasion d'un ajout de champ au sein de l'une des structures).

Initialement nous avions `string cat_anglais = malloc(strlen(cat_ang)+1)`; alors que `string cat_anglais = malloc(strlen(cat_ang)+2)`; était nécessaire.

```
210 /*: NAME : eng_cat ; traitement données anglais
211 ; ARGS : string english_part
212 ; USAGES : eng_cat(english_part)
213 ; GLOBALS : none
214 ; CALL : assoc_categorie
215 ; USER : div_eng_esp */
216
217 forme_anglais eng_cat(string english_part){
218
219     forme_anglais anglais = malloc(sizeof(struct Base));
220
221     string original_str = strdup(english_part);
222
223     /* traitement du mot anglais */
224     string mot_anglais = strtok(english_part, "{");
225     int len = strlen(mot_anglais); /* retire espace à la fin */
226     mot_anglais[len-1]='\0';
227
228     /* traitement de la catégorie anglaise */
229     string cat_ang = strtok(NULL, "");
230     /* travail sur les chaînes pour rajouter le { au début */
231     string debut = "{";
232     string cat_anglais = malloc(strlen(cat_ang)+2);
233     strcpy(cat_anglais, debut);
234     strcat(cat_anglais, cat_ang);
235
236     /* remplissage des éléments de struct [anglais] */
```

```

237     anglais->mot= strdup(mot_anglais);
238     anglais->original = strdup(original_str);
239     anglais->categorie=assoc_categorie(cat_anglais);
240
241     return anglais; /* renvoie la struct remplie */
242 }

```

La fonction spanish

La fonction `spanish` sépare les différentes traductions d'un mot en espagnol et les structures.

La fonction prend en paramètre une chaîne de caractères `esp_part` qui contient un ou plusieurs mots espagnols, éventuellement séparés par des virgules. Son deuxième argument est la chaîne de caractères `cat` qui représente la catégorie associée au mot anglais.

Tout au long de la fonction, les chaînes de caractères sont dupliquées afin d'éviter la modification des chaînes originales.

On initialise `premiere_trad` et `trad_prec` à `NULL`. `premiere_trad` sera renvoyée par la fonction.

La chaîne de caractères `esp_part` est dupliquée afin d'éviter la modification de la chaîne originale.

La fonction utilise `strtok_r` car la fonction appelée en son sein `subdiv_esp` utilise également `strtok` et cela causait des erreurs avec l'utilisation de la fonction `strtok`.

`strtok_r` est utilisée pour séparer les différentes traductions espagnoles.

Pour chaque traduction espagnole (ici `token`), on alloue de la mémoire pour `espagnol_trad` de type `trad_espagnol` (un pointeur vers `Trad`). En outre, on obtient la forme structurée de la traduction espagnole grâce à un appel à la fonction `subdiv_esp` qui prend en argument le `token` et la catégorie.

On enregistre cette forme dans le champ `espagnol` de `espagnol_trad`.

Par ailleurs, les traductions espagnoles sont connectées entre elles par leur `cdr`. C'est-à-dire qu'elles forment une liste.

La fonction renvoie les différentes traductions d'un mot complétées.

```

245 /* NAME : spanish ; traitement données traduction esp
246 ; ARGS : string esp_part, string cat
247 ; USAGES : spanish(esp_part, cat)
248 ; GLOBALS : none
249 ; CALL : subdiv_esp
250 ; USER : div_eng_esp */
251
252 trad_espagnol spanish(string esp_part, string cat){
253
254     trad_espagnol premiere_trad = NULL;
255     trad_espagnol trad_prec = NULL;
256
257     string token = NULL;
258
259     string copy_esp_part = strdup(esp_part);
260     char *ptr;
261     token = strtok_r(copy_esp_part, ",", &ptr);
262     /* strtok_r car subdiv_esp utilise aussi strtok */
263
264     while (token) {
265         trad_espagnol espagnol_trad = malloc(sizeof(struct Trad));
266         forme_espagnol traduction = subdiv_esp(token, cat);
267
268         espagnol_trad->espagnol= traduction;

```

```

269     espagnol_trad->cdr = NULL; /* initialisation à NULL pour éviter bugs! */
270
271     /* connexion des trads via le cdr */
272     if (premiere_trad == NULL){ /* si début de liste */
273         premiere_trad = espagnol_trad; /* 1er élément */
274     }
275     else { /* si pas début de la liste */
276         trad_prec->cdr=espagnol_trad; /* ajout comme dernier élt */
277     }
278     trad_prec = espagnol_trad; /* parcours des élt */
279     token= strtok_r(NULL, " ", &ptr); /* token suivant */
280 }
281 return premiere_trad; /* retour à partir du 1er élt */
282 }

```

La fonction `subdiv_esp`

La fonction `subdiv_esp` traite une chaîne de caractères qui contient la traduction d'un mot en espagnol et possiblement sa catégorie. Elle renvoie la forme en espagnol du mot complétée.

Une chaîne de caractères `esp_original` contient le mot en espagnol et potentiellement sa catégorie.

On alloue de la mémoire pour `espagnol`, une nouvelle `forme_espagnol` (un pointeur vers Base).

Tout au long de la fonction, les chaînes de caractères sont dupliquées afin d'éviter la modification des chaînes originales.

Le mot espagnol est extrait en utilisant `strtok` pour séparer la chaîne au caractère `{`. Ensuite, la chaîne de caractères correspondant au mot est traitée, notamment pour supprimer les espaces en début et en fin de chaîne.

On utilise `strtok` une seconde fois pour extraire la catégorie grammaticale du mot espagnol, si elle existe. Si aucune catégorie espagnole n'est spécifiée (c'est-à-dire si le résultat du second appel à `strtok` est `NULL`), alors on utilise la catégorie du mot anglais qui a été passée en argument à la fonction `subdiv_esp`.

En revanche, si une catégorie espagnole est fournie, alors elle est traitée pour supprimer l'espace en fin de chaîne et pour rajouter le `{` en début de chaîne.

Le mot espagnol, la chaîne originale (avant traitement, permettant la reconstruction ultérieure du dictionnaire) et la catégorie du mot (soit la catégorie espagnole traitée, soit celle du mot anglais utilisée) sont enregistrés dans les champs de `espagnol`. La catégorie est traitée grâce à un appel à la fonction `assoc_categorie`.

La fonction renvoie `espagnol` remplie avec les informations traitées.

```

285 /*; NAME : subdiv_esp ; traitement données forme esp
286 ; ARGS : string esp_original, string cat
287 ; USAGES : subdiv_esp(esp_original, cat)
288 ; GLOBALS : none
289 ; CALL : assoc_categorie
290 ; USER : spanish */
291
292 forme_espagnol subdiv_esp(string esp_original, string cat){
293
294     forme_espagnol espagnol = malloc(sizeof(struct Base));
295     string str_original = strdup(esp_original);
296     string esp_original_copy = strdup(str_original);
297
298     /* traitement du mot espagnol */
299     string tok = strtok(esp_original_copy, "{");
300     int len = strlen(tok);

```

```

301 tok[len]='\0'; /*suppression espace final */
302 string tok_bis = &tok[1]; /* suppression espace avant */
303
304 /* catégorie esp est le strktok suivant */
305 string tok2 = strtok(NULL, "");
306
307 if (tok2 == NULL){ /* s'il n'y a pas de catégorie esp */
308     espagnol->mot = tok_bis; /* le mot est le mot traité */
309     espagnol->original = strdup(str_original); /* trace de l'original */
310     espagnol->categorie = assoc_categorie(cat);
311     /* on utilise la catégorie du mot anglais */
312 }
313
314 else { /* s'il y a une catégorie du mot espagnol fournie */
315     /* traitement de la catégorie */
316     string cut_cat = tok2;
317     int len_cut = strlen(tok_bis);
318     tok_bis[len_cut-1]='\0'; /* supprimer dernier caractère */
319     string ma_cat = (malloc(strlen(cut_cat)+2));
320     ma_cat[0] = '\0'; /* initialisation */
321     strcat(ma_cat, "{"); /* ajout du { */
322     strcat(ma_cat, cut_cat); /* obtention cat complete */
323
324     /* on effectue les associations pour espagnol */
325     espagnol->mot = tok_bis;
326     espagnol->original = strdup(str_original);
327     espagnol->categorie = assoc_categorie(ma_cat);
328 }
329 return espagnol; /* on renvoie la struct formée */
330 }

```

Le fichier category.c

Le fichier `category.c` inclut les fichiers `sys.h` et `structure.h`.

```
1 /* # Nom ..... : category.c
2 # Rôle ..... : mäj des cat grammaticales des mots
3 # Auteur ..... : Avrië Floro
4 # Version ..... : V0.1 du 17/11/23
5 # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 # Usage : compilé via "make" ou "make -f makefile_tests" */
7
8 #include "sys.h"
9 #include "structure.h"
```

Le fichier `category.c` définit des macros qui sont utilisées pour assigner des valeurs numériques constantes à différents types de catégories grammaticales.

Chaque ligne définit une macro qui associe un nom de catégorie grammaticale (comme `NOUN`, `VERB`, `ADJECTIVE`, etc) à un nombre unique. Ces nombres servent d'identifiants uniques pour les catégories grammaticales. Ces macros permettent de pouvoir utiliser le nom des catégories de façon lisible plutôt que l'`int` correspondant.

```
11 #define NOUN 1
12 #define VERB 2
13 #define ADJECTIVE 3
14 #define ADVERB 4
15 #define INTERJECTION 5
16 #define PRONOUN 6
17 #define PREFIX 7
18 #define SUFFIX 8
19 #define PROVERB 9
20 #define PROPER_NOUN 10
21 #define PHRASE 11
22 #define PREP_PHRASE 12
23 #define CONJUNCTION 13
24 #define DETERMINER 14
25 #define CONTRACTION 15
26 #define ART 16
27 #define SYMBOL 17
28 #define CARDINAL_NUM 18
29 #define ACRONYM 19
30 #define PREPOSITION 20
31 #define ABBREVIATION 21
32 #define INITIALISM 22
33 #define PARTICLE 23
34 #define NUM 24
35 #define NOM_MASCULIN 25
36 #define NOM_FEMININ 26
37 #define NOM_PLURIEL 27
38 #define NOM_MASCULIN_PLURIEL 28
39 #define NOM_FEMININ_PLURIEL 29
40 #define NOM_EPICENE 30
41 #define SINGULIER 31
```

La fonction assoc_categorie

La fonction `assoc_categorie` associe une catégorie grammaticale présentée sous forme de chaîne de caractères à son équivalent numérique.

La fonction prend en paramètre une chaîne de caractères qui représente la catégorie grammaticale d'un mot (anglais ou espagnol).

La fonction alloue de la mémoire pour `cat`, de type `la_categorie` (un pointeur vers `Category`).

La chaîne de caractères passée en argument est dupliquée pour éviter la modification de la chaîne originale. La chaîne dupliquée est associée à `cat->chaîne`.

Grâce à des `if` et `else if`, on compare la chaîne de la catégorie traitée avec des chaînes prédéfinies (qui correspondent aux catégories présentes dans notre dictionnaire). Chaque catégorie grammaticale est associée à une valeur numérique définie au début du fichier. On associe la valeur numérique dans la structure via `cat->cat`.

La fonction renvoie `cat` complétée avec la chaîne de la catégorie et sa valeur numérique correspondante.

```
44 /*; NAME : assoc_categorie ; associe str cat et int du mot
45 ; ARGS : string cat_originale
46 ; USAGES : assoc_categorie(cat_originale)
47 ; GLOBALS : none
48 ; CALL : none
49 ; USER : eng_cat, subdiv_esp */
50
51 la_categorie assoc_categorie(string cat_originale){
52
53     la_categorie cat = (malloc(sizeof(struct Category)));
54     string cat_dup = strdup(cat_originale);
55     cat->chaîne = cat_dup; /* association cat->chaîne*/
56     /* définition du int de la catégorie */
57     if (strcmp(cat_dup, "{n}") == 0 || strcmp(cat_dup, "{n} ") == 0) {
58         cat->cat = NOUN;
59     }
60     else if (strcmp(cat_dup, "{v}") == 0) {
61         cat->cat = VERB;
62     }
63     else if (strcmp(cat_dup, "{adj}") == 0) {
64         cat->cat = ADJECTIVE;
65     }
66     else if (strcmp(cat_dup, "{adv}") == 0) {
67         cat->cat = ADVERB;
68     }
69     else if (strcmp(cat_dup, "{interj}") == 0) {
70         cat->cat = INTERJECTION;
71     }
72     else if (strcmp(cat_dup, "{pron}") == 0) {
73         cat->cat = PRONOUN;
74     }
75     else if (strcmp(cat_dup, "{prefix}") == 0) {
76         cat->cat = PRÉFIX;
77     }
78     else if (strcmp(cat_dup, "{suffix}") == 0) {
79         cat->cat = SUFFIX;
80     }
81     else if (strcmp(cat_dup, "{proverb}") == 0) {
82         cat->cat = PROVERB;
83     }
84     else if (strcmp(cat_dup, "{prop}") == 0) {
85         cat->cat = PROPER_NOUN;
86     }
87     else if (strcmp(cat_dup, "{phrase}") == 0) {
88         cat->cat = PHRASE;
89     }
```



```

90     else if (strcmp(cat_dup, "{prep phrase}") == 0) {
91         cat->cat = PREP_PHRASE;
92     }
93     else if (strcmp(cat_dup, "{conj}") == 0) {
94         cat->cat = CONJUNCTION;
95     }
96     else if (strcmp(cat_dup, "{determiner}") == 0) {
97         cat->cat = DETERMINER;
98     }
99     else if (strcmp(cat_dup, "{contraction}") == 0) {
100         cat->cat = CONTRACTION;
101     }
102     else if (strcmp(cat_dup, "{art}") == 0) {
103         cat->cat = ART;
104     }
105     else if (strcmp(cat_dup, "{symbol}") == 0) {
106         cat->cat = SYMBOL;
107     }
108     else if (strcmp(cat_dup, "{cardinal num}") == 0) {
109         cat->cat = CARDINAL_NUM;
110     }
111     else if (strcmp(cat_dup, "{acronym}") == 0) {
112         cat->cat = ACRONYM;
113     }
114     else if (strcmp(cat_dup, "{prep}") == 0) {
115         cat->cat = PRÉPOSITION;
116     }
117     else if (strcmp(cat_dup, "{abbr}") == 0) {
118         cat->cat = ABBREVIATION;
119     }
120     else if (strcmp(cat_dup, "{initialism}") == 0) {
121         cat->cat = INITIALISM;
122     }
123     else if (strcmp(cat_dup, "{particle}") == 0) {
124         cat->cat = PARTICLE;
125     }
126     else if (strcmp(cat_dup, "{num}") == 0) {
127         cat->cat = NUM;
128     }
129     else if (strcmp(cat_dup, "{m}") == 0) {
130         cat->cat = NOM_MASCULIN;
131     }
132     else if (strcmp(cat_dup, "{f}") == 0) {
133         cat->cat = NOM_FEMININ;
134     }
135     else if (strcmp(cat_dup, "{p}") == 0) {
136         cat->cat = NOM_PLURIEL;
137     }
138     else if (strcmp(cat_dup, "{m.p}") == 0) {
139         cat->cat = NOM_MASCULIN_PLURIEL;
140     }
141     else if (strcmp(cat_dup, "{f.p}") == 0) {
142         cat->cat = NOM_FEMININ_PLURIEL;
143     }
144     else if (strcmp(cat_dup, "{f}{m}") == 0 || strcmp(cat_dup, "{f}/{m}") == 0 ||
145             strcmp(cat_dup, "{m}{f}") == 0 || strcmp(cat_dup, "{m}/{f}") == 0){
146         cat->cat = NOM_EPICENE;
147     }
148     else if (strcmp(cat_dup, "{s}") == 0) {
149         cat->cat = SINGULIER;
150     }

```

```
151  
152  
153 return cat; /* renvoie la catégorie */
```

Le fichier test.c

Le fichier `test.c` inclut les fichiers `sys.h` et `structure.h`.

Ce fichier est utilisé lors de la compilation, uniquement lorsque l'on souhaite lancer les tests de reconstruction de notre programme. C'est-à-dire lorsque l'on lance la compilation avec le Makefile `makefile_tests` qui s'exécute avec la commande `make -f makefile_tests`.

Ce fichier n'est pas utilisé lorsque notre programme est lancé pour servir de dictionnaire bilingue en ligne de commande.

```
1 /* # Nom ..... : test.c
2 # Rôle ..... : initialise les fichiers et lance les tests
3 # Auteur ..... : Avrile Floro
4 # Version ..... : V0.1 du 17/11/23
5 # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 # Usage : compilé seulement pour réaliser les tests
7 # Usage : pour les tests de reconstruction > make -f makefile_tests
8 #      puis > ./dictionnaire_test */
9
10
11 #include "sys.h"
12 #include "structure.h"
```

La fonction initialisation_fichiers

La fonction `initialisation_fichiers` permet d'initialiser les fichiers qui seront utilisés pour les tests. Les fichiers sont d'abord ouverts en écriture (`w`) pour supprimer leur contenu précédent (s'il existait). Les fichiers sont ensuite fermés.

Cela sert uniquement à pouvoir réinitialiser les fichiers lors de chaque lancement du programme, de façon à ce que cela ne cause pas de problèmes en cas de lancements multiples du programme sans nettoyage préalable des fichiers créés (avec l'option `make -f makefile_tests clean`).

```
15 /*; NAME : initialisation_fichier(); init fichier en W si -DTEST
16 ; ARGS :
17 ; USAGES : initialisation_fichier()
18 ; GLOBALS : none
19 ; CALL : none
20 ; USER : main */
21
22 void initialisation_fichiers(){
23     /* initialisation des fichiers en W pour effacer les contenus précédents */
24     /* fermeture */
25     FILE *_mot_anglais = fopen("mot_anglais.txt", "w");
26     fclose(_mot_anglais);
27     FILE *_all_words_traduc_esp = fopen("all_words_traduc_esp.txt", "w");
28     fclose(_all_words_traduc_esp);
29     FILE *_all_trad_esp_treated = fopen("all_trad_esp_treated.txt", "w");
30     fclose(_all_trad_esp_treated);
31     FILE *_cat_anglais = fopen("cat_mot_anglais.txt", "w");
32     fclose(_cat_anglais);
33     FILE *_cat_esp = fopen("cat_mot_esp.txt", "w");
34     fclose(_cat_esp);
35     FILE *_reconstruction = fopen("reconstruction.txt", "w");
36     fclose(_reconstruction);
37     FILE *_anglais = fopen("tout_anglais.txt", "w");
38     fclose(_anglais);
39     FILE *_espagnol = fopen("trad_espagnol.txt", "w");
40     fclose(_espagnol);
```

La fonction `creation_test`

On utilise un pointeur `p_dico` afin de ne pas modifier les données originales lors de notre parcourt du dictionnaire.

On parcourt les différentes entrées du dictionnaire, et pour chaque entrée on ouvre les fichiers en mode append `a`. Cette fois, nous souhaitons que les données s'écrivent sans effacer les données précédentes.

Lors du parcourt des entrées du dictionnaire, on peut remplir les fichiers qui contiennent les mots anglais, les catégories des mots anglais, les mots anglais et leur catégorie, et une partie du fichier de reconstruction.

Ensuite, on va parcourir les différentes traductions d'un mot en utilisant encore une fois un pointeur afin de ne pas modifier les données originales.

Lors du parcourt des traductions, on ouvre les fichiers nécessaires en mode append `a`. On peut imprimer dans les fichiers le contenu qui correspond à tous les mots espagnols, aux traductions espagnoles traitées, aux catégories des mots espagnol et aux mots espagnols et à leur possible catégorie originale. On continue de compléter notre fichier de reconstruction grâce au champ `original`.

Après chaque ouverture des fichiers, nous les refermons. Puisque nos fichiers ont été précédemment déclarés en `w` puis ensuite en `a`, cela nous coûte de devoir les refermer après chaque utilisation pour ne pas avoir d'erreurs.

On parcourt les traductions des mots et les entrées du dictionnaires.

```

44 /*; NAME : creation_tests ; lance les tets vers fichiers en A
45 ; ARGS : dictionnaire
46 ; USAGES : creation_tests(dictionnaire)
47 ; GLOBALS : none
48 ; CALL : none
49 ; USER : main */
50
51 void creation_tests(une_entree dictionnaire){
52
53     une_entree p_dico = dictionnaire; /* ptr pour pas modif original */
54
55     while (p_dico != NULL){
56         /* ouverture fichiers en mode append pour ajouter à chaque boucle */
57         /* fermeture des fichiers */
58         FILE *f_mot_anglais = fopen("mot_anglais.txt", "a");
59         fprintf(f_mot_anglais, "%s\n", p_dico->anglais->mot);
60         fclose(f_mot_anglais);
61         FILE *f_cat_anglais = fopen("cat_mot_anglais.txt", "a");
62         fprintf(f_cat_anglais, "%s\n", p_dico->anglais->categorie->chaine);
63         fclose(f_cat_anglais);
64         FILE *f_anglais = fopen("tout_anglais.txt", "a");
65         fprintf(f_anglais, "%s\n", p_dico->anglais->original);
66         fclose(f_anglais);
67         FILE *f_reconstruction = fopen("reconstruction.txt", "a");
68         fprintf(f_reconstruction, "%s", p_dico->anglais->original);
69         fclose(f_reconstruction);
70
71         trad_espagnol temp = p_dico->traduction; /* ptr pour pas altérer original */
72         while(temp != NULL){ /* parcourt des trads */
73
74             /* ouverture en mode append */
75             FILE *f_all_words_traduc_esp = fopen("all_words_traduc_esp.txt", "a");
76             FILE *f_all_trad_esp_treated = fopen("all_trad_esp_treated.txt", "a");
77             FILE *f_cat_esp = fopen("cat_mot_esp.txt", "a");

```

```

78     FILE *f_espagnol = fopen("trad_espagnol.txt", "a");
79     f_reconstruction = fopen("reconstruction.txt", "a");
80
81     /* impression dans les fichiers */
82     fprintf(f_all_words_traduc_esp, "%s\n", temp->espagnol->mot);
83     fprintf(f_all_trad_esp_treated, "%s %s\n", temp->espagnol->mot,
temp->espagnol->categorie->chaine);
84     fprintf(f_cat_esp, "%s\n", temp->espagnol->categorie->chaine);
85     fprintf(f_reconstruction, "%s", temp->espagnol->original);
86     fprintf(f_espagnol, "%s", temp->espagnol->original);
87
88     /* fermeture des fichiers */
89     fclose(f_all_words_traduc_esp);
90     fclose(f_all_trad_esp_treated);
91     fclose(f_cat_esp);
92     fclose(f_reconstruction);
93     fclose(f_espagnol);
94
95     if (temp->cdr != NULL){
96         f_reconstruction = fopen("reconstruction.txt", "a");
97         f_espagnol = fopen("trad_espagnol.txt", "a");
98         fprintf(f_reconstruction, ",");
99         fprintf(f_espagnol, ",");
100         fclose(f_reconstruction);
101         fclose(f_espagnol);
102     }
103
104     temp = temp->cdr; /* parcourt des trads du dico */
105 }
106 if (p_dico->cdr != NULL){
107     f_reconstruction = fopen("reconstruction.txt", "a");
108     FILE *f_espagnol = fopen("trad_espagnol.txt", "a");
109     fprintf(f_reconstruction, "\n");
110     fprintf(f_espagnol, "\n");
111     fclose(f_reconstruction);
112     fclose(f_espagnol);
113 }
114 p_dico = p_dico->cdr; /* parcourt des entrées du dico */
115 }
116 }

```

1.4 Programme en C - Le lancement des tests

Selon les directives de compilation utilisées, notre programme sert de dictionnaire bilingue en ligne de commande ou lance des tests. Nous avons décidé de séparer ces deux fonctionnalités afin de préserver les performances de notre programme en ligne de commande.

Afin de lancer les test, nous avons prévu un Makefile nommé `makefile_tests`. Pour exécuter ce Makefile, il convient de lancer la commande `make -f makefile_tests`. Les directives de compilation sont `gcc -g -DTEST -Wall dictionnaire.c test.c category.c -o dictionnaire_test`.

Puisque l'on veut lancer les tests, on ajoute aux direction de compilation le fichier `test.c`.

L'exécutable créé est alors `dictionnaire_test` qu'il convient de lancer grâce à la commande `./dictionnaire_test`.

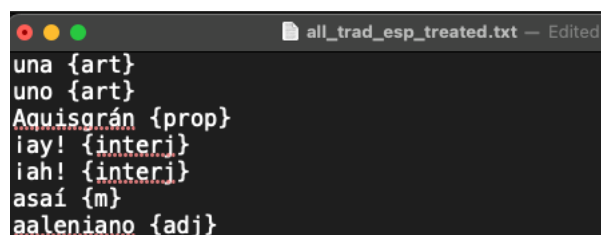
On peut nettoyer les fichier obtenus grâce à la commande `make -f makefile_tests clean` mais cela n'est pas nécessaire pour le bon fonctionnement des fichiers puisqu'ils sont réinitialisés lors de chaque exécution du programme en test. Cela permet néanmoins de gagner en visibilité.

Voici l'intégralité du Makefile créé afin de lancer les tests. Comme on peut le constater, les directives de compilation incluent `-DTEST`. Nous nous y référons tout au long de notre programme grâce à `# ifdef TEST`.

```
1 # Nom ..... : makefile_tests
2 # Rôle ..... : compile dictionnaire_test
3 # Auteur ..... : AvriLe Floro
4 # Version ..... : V0.1 du 17/11/23
5 # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 # Usage : make -f makefile_tests
7
8 all: compile
9
10 compile:
11 gcc -g -DTEST -Wall dictionnaire.c test.c category.c -o dictionnaire_test
12
13 clean:
14 rm -f dictionnaire_test
15 rm -f all_trad_esp_treated.txt
16 rm -f all_words_traduc_esp.txt
17 rm -f cat_mot_anglais.txt
18 rm -f cat_mot_esp.txt
19 rm -f mot_anglais.txt
20 rm -f reconstruction.txt
21 rm -f tout_anglais.txt
22 rm -f trad_espagnol.txt
```

Le lancement des tests engendre la création de huit fichiers distincts.

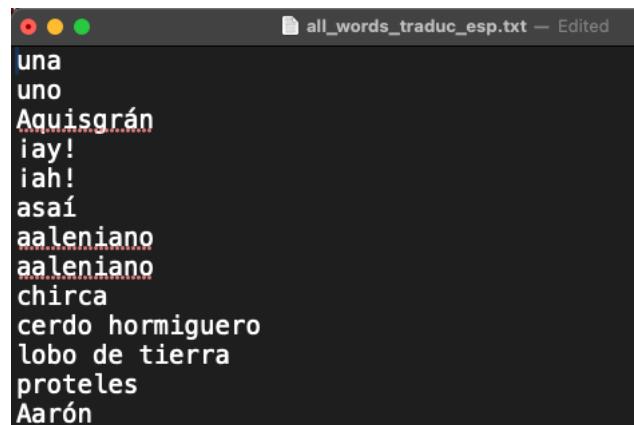
- `all_trad_esp_treated.txt` : Ce fichier affiche toutes les traductions espagnoles traitées, c'est-à-dire avec une catégorie associée. Ce fichier est initialisé grâce à `initialisation_fichier` puis construit grâce à `creation_tests`.



```
una {art}
uno {art}
Aquisgrán {prop}
¡ay! {interj}
¡ah! {interj}
así {m}
aaleniano {adj}
```

FIGURE 18 – Aperçu du fichier "all_trad_esp_treated.txt".

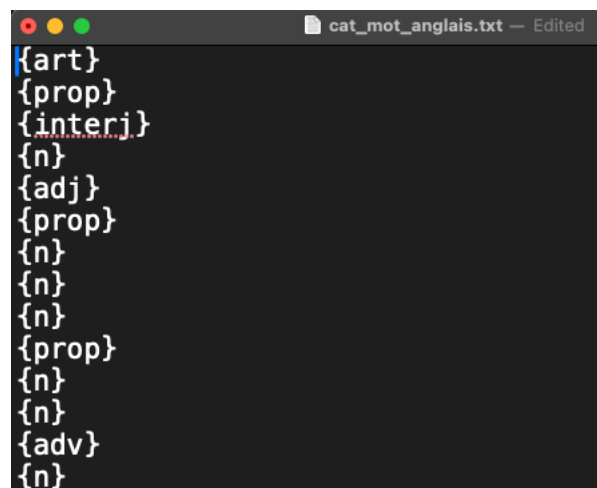
- `all_words_traduc_esp.txt` : Ce fichier affiche toutes les traductions espagnoles (sans catégorie). Ce fichier est initialisé grâce à `initialisation_fichier` puis construit grâce à `creation_tests`.



```
una
uno
Aquisgrán
iaay!
iah!
asaí
aaleniano
aaleniano
chirca
cerdo hormiguero
lobo de tierra
proteles
Aarón
```

FIGURE 19 – Aperçu du fichier "`all_words_traduc_esp.txt`".

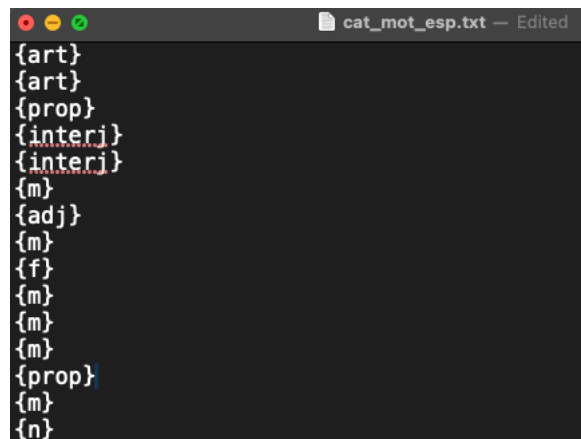
- `cat_mot_anglais.txt` : Ce fichier affiche les catégories des mots anglais. Ce fichier est initialisé grâce à `initialisation_fichier` puis construit grâce à `creation_tests`.



```
{art}
{prop}
{interj}
{n}
{adj}
{prop}
{n}
{n}
{n}
{n}
{prop}
{n}
{n}
{adv}
{n}
```

FIGURE 20 – Aperçu du fichier "`cat_mot_anglais.txt`".

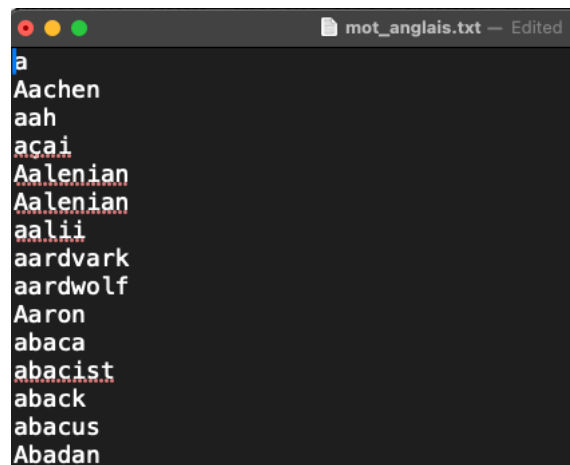
- `cat_mot_esp.txt` : Ce fichier affiche les catégories de toutes les traductions espagnoles des mots. Ce fichier est initialisé grâce à `initialisation_fichier` puis construit grâce à `creation_tests`.



```
{art}
{art}
{prop}
{interj}
{interj}
{m}
{adj}
{m}
{f}
{m}
{m}
{m}
{prop}
{m}
{n}
```

FIGURE 21 – Aperçu du fichier "cat_mot_esp.txt".

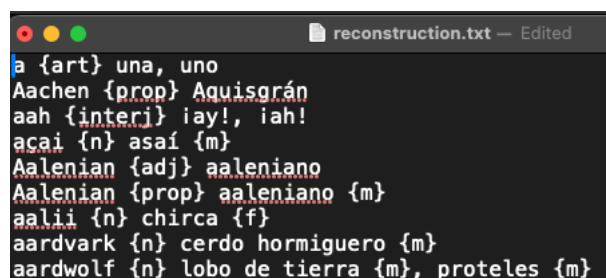
- `mot_anglais.txt` : Ce fichier affiche tous les mots anglais (sans catégorie). Ce fichier est initialisé grâce à `initialisation_fichier` puis construit grâce à `creation_tests`.



```
a
Aachen
aah
acai
Aalenian
Aalenian
aalii
aardvark
aardwolf
Aaron
abaca
abacist
aback
abacus
Abadan
```

FIGURE 22 – Aperçu du fichier "mots_anglais.txt".

- `reconstruction.txt` : Ce fichier reconstruit le dictionnaire `mon_dico.txt` à l'original à partir des champs `original` du mot anglais et des traductions. `reconstruction.txt` est le fruit d'une reconstruction complète des données. Ce fichier est initialisé grâce à `initialisation_fichier` puis construit grâce à `creation_tests`.



```
a {art} una, uno
Aachen {prop} Aquisgrán
aah {interj} iay!, iah!
acai {n} asai {m}
Aalenian {adj} aaleniano
Aalenian {prop} aaleniano {m}
aalii {n} circa {f}
aardvark {n} cerdo hormiguero {m}
aardwolf {n} lobo de tierra {m}, proteles {m}
```

FIGURE 23 – Aperçu du fichier "reconstruction.txt".

Grâce à la commande `diff mon_dico.txt reconstruction.txt`, on constate que le fichier `reconstruction.txt` et le fichier `mon_dico.txt` sont identiques. Nous avons lancé l'opération sous Mac et sous Linux. Les résultats sont les mêmes.

```
→ tests diff mon_dico.txt reconstruction.txt
→ tests
```

FIGURE 24 – Le fichier "reconstruction.txt" est identique au fichier "mon_dico.txt" (test sous Mac).

```
avrile@avrile: ~/Desktop/6.Listes 6 Dico/2_prog_C
avrile@avrile:~/Desktop/6.Listes 6 Dico/2_prog_C$ make -f makefile_tests
gcc -g -DTEST -Wall dictionnaire.c test.c category.c -o dictionnaire_test
avrile@avrile:~/Desktop/6.Listes 6 Dico/2_prog_C$ ./dictionnaire_test
avrile@avrile:~/Desktop/6.Listes 6 Dico/2_prog_C$ diff mon_dico.txt reconstruction.txt
avrile@avrile:~/Desktop/6.Listes 6 Dico/2_prog_C$
```

FIGURE 25 – Le fichier "reconstruction.txt" est identique au fichier "mon_dico.txt" (test sous Linux).

- `tout_anglais.txt` : Ce fichier affiche tous les mots anglais et leur catégorie. Ce fichier est initialisé grâce à `initialisation_fichier` puis construit grâce à `creation_tests`.

```
tout_anglais.txt — Edited
a {art}
Aachen {prop}
aah {interi}
acai {n}
Aalenian {adj}
Aalenian {prop}
aalii {n}
aardvark {n}
aardwolf {n}
Aaron {prop}
abaca {n}
abacist {n}
aback {adv}
```

FIGURE 26 – Aperçu du fichier "tout_anglais.txt".

- `trad_espagnol.txt` : Ce fichier affiche toutes les traductions espagnoles d'un mot, suivies éventuellement d'une catégorie. Ce fichier est initialisé grâce à `initialisation_fichier` puis construit grâce à `creation_tests`.

```
trad_espagnol.txt — Edited
una, uno
Aquisgrán
iay!, iah!
asaí {m}
aaleniano
aaleniano {m}
chirca {f}
cerdo hormiguero {m}
lobo de tierra {m}, proteles {m}
```

FIGURE 27 – Aperçu du fichier "trad_espagnol.txt".

1.5 Programme en C - Exemple d'utilisation du dictionnaire bilingue

Nous avons préparé un Makefile nommé **Makefile**, permettant de faciliter la compilation de notre programme lorsque nous voulons utiliser le dictionnaire bilingue en ligne de commande.

Les directives de compilation utilisées par le Makefile sont `gcc -Wall dictionnaire.c category.c -o dictionnaire`. On obtient suite à l'exécution du Makefile un exécutable **dictionnaire** que l'on peut lancer grâce à la commande `./dictionnaire`.

On peut nettoyer les fichiers créés par le Makefile avec la commande `make clean`. Voici l'intégralité du Makefile :

```
1 # Nom ..... : makefile
2 # Rôle ..... : compile dictionnaire
3 # Auteur ..... : Avrile Floro
4 # Version ..... : V0.1 du 17/11/23
5 # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 # Usage : make
7
8
9 all: compile
10
11 compile:
12     gcc -Wall dictionnaire.c category.c -o dictionnaire
13
14 clean:
15     rm -f dictionnaire
```

Nous illustrons notre programme en cherchant la traduction du verbe **smell**. Nous faisons également une recherche incorrecte (sans préciser la catégorie du nom). Le programme nous envoie **usage**. Nous continuons la traduction de deux noms (**computer** et **dog**). Enfin, nous faisons une recherche infructueuse.

Par ailleurs, nous avons compilé et testé notre programme, aussi bien sous Mac que sous Linux. Le programme compile sans erreur et s'exécute comme attendu sous les deux OS.

```
→ programme_c make
gcc -Wall dictionnaire.c category.c -o dictionnaire
→ programme_c ./dictionnaire
Quel mot voulez-vous traduire ?
smell {v}
husmear {v}, oler {v}, oler a {v}

Quel mot voulez-vous traduire ?
dog
Usage: mot_anglais {catégorie_recherché}

Quel mot voulez-vous traduire ?
dog {n}
perro {m}

Quel mot voulez-vous traduire ?
computer {n}
computadora {f}, ordenador {m}, computador {m}

Quel mot voulez-vous traduire ?
inventé {num}
Aucune traduction n'a été trouvée.

Quel mot voulez-vous traduire ?
```

FIGURE 28 – Des exemples d'utilisation de "./dictionnaire" (test sous Mac).

```
avrile@avrile:~/Desktop/6.Listes 6 Dico/2_prog_C$ make
gcc -Wall dictionnaire.c category.c -o dictionnaire
avrile@avrile:~/Desktop/6.Listes 6 Dico/2_prog_C$ ./dictionnaire
Quel mot voulez-vous traduire ?
a {art}
uno {art}, una {art}

Quel mot voulez-vous traduire ?
hundred {num}
ciento {num}, cien {num}

Quel mot voulez-vous traduire ?
cry {v}
llorar {v}

Quel mot voulez-vous traduire ?
noseque {n}
Aucune traduction n'a été trouvée.

Quel mot voulez-vous traduire ?
onverrabien
Usage: mot_anglais {catégorie_recherché}
```

FIGURE 29 – Des exemples d'utilisation de "./dictionnaire" (test sous Linux).

Annexes

.1 Le code source "prototype.lisp"

Voici l'intégralité du code source `prototype.lisp`.

```
1 ; Nom ..... : prototype.lisp
2 ; Role ..... : prototype du dico en lisp
3 ; Auteur ..... : Avrië Floro
4 ; Version ..... : V0.1 du 22/12/23
5 ; Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 ; Usage : lisp puis (load "prototype.lisp") puis (traduction 'love '{v})
7
8 ; NAME : lire-fichier ; divise le dico en liste de lignes
9 ; ARGS : fichier en entrée
10 ; USAGES : (lire-fichier fichier)
11 ; GLOBALS : none
12 ; CALL : none
13 ; USER : main
14
15 (defun lire-fichier (nom_fichier)
16   (with-open-file (fichier nom_fichier)
17     (loop for ligne = (read-line fichier nil) ; lit une ligne
18       while ligne ; tant qu'il y a une ligne
19       collect (list ligne) ) ) ; chaque sous liste contient une ligne
20
21 ; NAME : etape1 ; le fichier sous forme de liste de listes
22 ; USAGES : appel de la fonction lire-fichier > (("aback {adv} hacia atrás")
23 ; ("another day, another dollar {proverb}
24 ; USER :
25
26 ; (setq etape1 (lire-fichier "mon_dico.txt"))
27
28 ; NAME : div-english ; isole la partie anglaise de la ligne dans une liste
29 ; ARGS : chaîne &aux index-fermeture english
30 ; USAGES : (div-english chaîne) > ("aback {adv}")
31 ; GLOBALS : none
32 ; CALL : none
33 ; USER : parentheses
34
35 (defun div-english (chaîne &aux index-fermeture english)
36   (setq index-fermeture (position #\} (car chaîne))) ; cherche la position de la fermeture
37   (setq english (string-trim '#\Space) (subseq (car chaîne) 0 (1+ index-fermeture))))
38   ; isoler la partie anglaise
39   (list english) ) ; renvoie la partie anglaise dans une liste
40
41
42 ; NAME : eng-word ; isole le mot anglais dans une liste
43 ; depuis la catégorie anglais
44 ; ARGS : chaîne &aux index-ouverture mot
45 ; USAGES : (eng-word chaîne) > ("aback")
46 ; GLOBALS : none
47 ; CALL : none
48 ; USER : parentheses
49
50 (defun eng-word (chaîne &aux index-ouverture mot)
51   (setq index-ouverture (position #\{ (car chaîne))) ; cherche la position de l'ouverture
52   (setq mot (string-trim '#\Space) (subseq (car chaîne) 0 index-ouverture)))
53   ; isole le mot anglais (avant l'ouverture)
```

```

54 (list mot) ) ; renvoie le mot anglais dans une liste
55
56
57 ; NAME : cat-word ; isole la catégorie du mot anglais
58 ; dans une liste depuis la catégorie anglais
59 ; ARGS : chaine &aux index-ouverture index-fermeture mot
60 ; USAGES : (eng-word chaine) > ("{adv}")
61 ; GLOBALS : none
62 ; CALL : none
63 ; USER : parentheses
64
65 (defun cat-word (chaine &aux index-ouverture index-fermeture mot)
66 (setq index-ouverture (position #\{ (car chaine))) ; cherche la position de l'ouverture
67 (setq index-fermeture (position #\} (car chaine))) ; cherche la position de la fermeture
68 (setq mot (string-trim '(#\Space) (subseq (car chaine) index-ouverture (1+ index-fermeture))))
69 ; isole la catégorie du mot anglais (entre les deux)
70 (list mot) ) ; renvoie la catégorie du mot anglais dans une liste
71
72
73
74 ; NAME : div-spanish ; isole la partie espagnole de la ligne dans une liste
75 ; ARGS : chaine &aux index-fermeture spanish
76 ; USAGES : (eng-word chaine) > ("hacia atrás")
77 ; GLOBALS : none
78 ; CALL : none
79 ; USER : parentheses
80
81 (defun div-spanish (chaine &aux index-fermeture spanish)
82 (setq index-fermeture (position #\} (car chaine))) ; cherche la position de la fermeture
83 (setq spanish (string-trim '(#\Space) (subseq (car chaine) (1+ index-fermeture))))
84 ; isole la partie espagnole (après la fermeture)
85 (list spanish) ) ; renvoie la partie espagnole dans une liste
86
87 ; NAME : parentheses ; fais le parenthésage facile (grâce aux sous fonctions)
88 ; ARGS : liste &aux resultat engpart engmot catmot spanish ligne
89 ; USAGES : (parentheses etape1) > (((("aback") ("{adv}")) ("hacia atrás")))
90 ; (((("another day, another dollar") ("{proverb}")) ("un día más un día menos")))
91 ; ("un día más un día menos") (((("hyper-"))
92 ; GLOBALS : none
93 ; CALL : div-english, eng-word, cat-word, div-spanish
94 ; USER : main
95
96 (defun parentheses (liste &aux resultat engpart engmot catmot spanish ligne)
97 (loop for item in liste do
98 (setq engpart (div-english item)) ; partie anglaise
99 (setq engmot (eng-word engpart)) ; mot anglais
100 (setq catmot (cat-word engpart)) ; catégorie du mot anglais
101 (setq spanish (div-spanish item)) ; partie espagnole
102 (setq ligne (list (list engmot catmot) spanish)) ; rassemble le tout dans la ligne
103 (push ligne resultat) )
104 (nreverse resultat) )
105
106
107
108
109 ; NAME : etape2 ; on applique à etape1 le parenthésage facile
110 ; USAGES : appel de la f. parenthess > (((("aback") ("{adv}")) ("hacia atrás"));
111 ; (((("another day, another dollar") ("{proverb}")) ("un día más un día menos")))
112 ; (((("hyper-"))
113 ; USER :
114 ; (setq etape2 (parentheses etape1))

```

```

115
116
117 ; NAME : split-by-one-comma ; sépare une chaîne de caractères en deux sous liste
118 ;      à la virgule utiliser pour séparer les différentes
119 ;      traductions possibles
120 ; ARGS : string
121 ; USAGES : (split-by-one-comma "a, v") > ("a" " v")
122 ; GLOBALS : none
123 ; CALL :
124 ; USER : isoler_diff_traduc
125 ; SOURCE: https://cl-cookbook.sourceforge.net/strings.html
126
127 * (defun split-by-one-comma (string)
128   (loop for i = 0 then (1+ j)
129     as j = (position #\, string :start i)
130     collect (subseq string i j)
131     while j) )
132
133 ; NAME : isoler_diff_traduc ; appelle split-by-one-comma sur la partie espagnole de la liste
134 ;      donne une sous liste pour chaque traduction possible
135 ; ARGS : liste (avec le full parenthésage)
136 ; USAGES : (isoler_diff_traduc (L)) > ("hacia atrás {v}" " hacia detrás {adv}")
137 ; GLOBALS : none
138 ; CALL :
139 ; USER : structure_en_cours
140
141 (defun isoler_diff_traduc (L)
142   (if (atom L) ; si fin de la liste
143     nil ; nil
144     (split-by-one-comma (caadr L)) ) )
145   ; sinon on appelle split-by-one-comma sur la partie espagnole de la liste
146
147 ; NAME : sous_liste ; crée des sous_liste à partir des éléments de la liste
148 ; ARGS : L &aux result
149 ; USAGES : (sous_liste '(a b c)) > ((a) (b) (c))
150 ; GLOBALS : none
151 ; CALL :
152 ; USER : structure_en_cours
153
154 (defun sous_liste (L &aux result)
155   (if (atom L)
156     nil ; Si L est un atome, renvoie nil
157     (loop for item in L do (push (list item) result)) )
158   ; Pour chaque élément de L, ajoute une sous-liste à result
159   (nreverse result) ) ; Renvoie result en ordre inversé
160
161
162
163 ; NAME : structure_en_cours ; crée un doublet pour le mot anglais et sa catégorie
164 ;      et crée des sous-listes pour chaque traduction possible >> il faudra
165 ;      encore isoler les catégories de l'espagnol
166 ; ARGS : liste &aux result prep new-item
167 ; USAGES : (sous_liste '(a b c)) > ((a) (b) (c))
168 ; GLOBALS : none
169 ; CALL : sous_liste, isoler_diff_traduc
170 ; USER : main
171
172 (defun structure_en_cours (liste &aux result prep new-item)
173   (dolist (item (reverse liste) result) ; pour chaque élément de la liste
174     (setq prep (isoler_diff_traduc item))
175     ; on sépare en sous-listes les différentes traductions du mot

```

```

176 (setq new-item (cons (cons (caaar item) (caadar item)) (list (sous_liste prep))))
177 ; on crée un doublet pour le mot anglais et sa catégorie
178 ; on crée des sous-listes pour chaque traduction possible
179 (push new-item result) ) ) ; on ajoute à la liste résultat
180
181
182 ; NAME : etape3 ; on applique à etape2 la construction en doublet pour l'anglais
183 ; et les sous-liste de traduction pour l'espagnol
184 ; USAGES : (("awful" . "{adj}") ("horrible {adj}") ("espantoso {adj}") ("terrible {adj}")
185 ; ("tremendo {adj}"))
186 ; USER :
187 ; (setq etape3 (structure_en_cours etape2))
188
189
190 ; NAME : process-sublist ; divise une sous liste en > la traduction espagnole et sa catégorie
191 ; et construit les doublets
192 ; ARGS : liste liste1bis &aux index-ouverture index-fermeture cat
193 ; USAGES : (("llorón {m}") ("quejumbroso {m}")) > ("llorón" . "{m}")
194 ; GLOBALS : none
195 ; CALL :
196 ; USER : process-list
197
198 (defun process-sublist (liste liste1bis &aux index-ouverture index-fermeture cat)
199 (if (position #\{ (car liste)) ; S'il y a une catégorie pour le mot espagnol
200 (progn ; on retournera le résultat du cons
201 (setq index-ouverture (position #\{ (car liste))) ; Définit l'ouverture
202 (setq index-fermeture (position #\} (car liste))) ; Définit la fermeture
203 (setq cat (string-trim '(#\Space) (subseq (car liste) index-ouverture (1+ index-fermeture))))
204 ; Isole la catégorie de la traduction espagnole
205 (cons (string-trim '(#\Space) (subseq (car liste) 0 index-ouverture)) cat) )
206 ; Met sur un doublet le mot et sa catégorie
207 (cons (string-trim '(#\Space) (subseq (car liste) 0)) (cdr liste1bis)) ) )
208 ; Sinon, doublet avec mot espagnol et catégorie issue de l'anglais via liste1bis
209
210
211
212
213 ; NAME : process-list ; applique process-sublist à une liste de sous-liste
214 ; ARGS : liste liste1bis &aux (resultat nil)
215 ; USAGES : (("llorón {m}") ("quejumbroso {m}")) > ("llorón" . "{m}") ("quejumbroso" . "{m}")
216 ; GLOBALS : none
217 ; CALL : process-sublist
218 ; USER : forme_finale
219
220 (defun process-list (liste liste1bis &aux resultat)
221 (loop for element in liste do ; pour chaque elt
222 (push (process-sublist element (car liste1bis)) resultat) )
223 ; on ajoute le résultat de process-sublist à la liste résultat
224 (nreverse resultat) ) ; on renvoie la liste résultat inversée
225
226
227
228
229 ; NAME : forme_finale ; finit la construction de la liste en doublet pour l'anglais et pour l'espagnol
230 ; ARGS : liste liste1bis &aux res
231 ; USAGES : (forme_finale etape3 etape3)
232 ; GLOBALS :
233 ; CALL : process-list
234 ; USER : main
235
236 (defun forme_finale (liste liste1bis &aux res)

```



```

237 (dolist (elt liste res) ; pour chaque élément de la liste
238   (setq res
239     (cons ; on ajoute le mot traité à la liste résultat
240       (cons ; pour l'anglais + pour l'espagnol
241         (car elt)
242         (list (reverse (process-list (cadr elt) (car liste1bis))))))
243       res) )
244   (setq liste1bis (cdr liste1bis)) ) ; on itère sur la suite de liste1bis
245 (reverse res) ) ; on renvoie la liste résultat inversée
246
247
248
249
250 ; NAME : etape4 ; on applique à etape3 la construction en doublet pour les sous listes de l'espagnol
251 ; USAGES : (("bellyacher" . "{n}") (("llorón" . "{m}") ("quejumbroso" . "{m}")))
252 ; USER :
253 ; (setq etape4 (forme_finale etape3 etape3))
254
255
256 ; NAME : imprimer-anglais ; renvoie que les mots en anglais (on utilise etape2
257 ; car à partir de etape3 on a modifié la construction du dico > en vue des comparaisons)
258 ; ARGS : liste
259 ; USAGES : (imprimer-anglais etape2) ("aback" "another day, another dollar" "hyper-" "-ic" "Majesty"
    "Mandarin"...
260 ; GLOBALS : none
261 ; CALL : none
262 ; USER : printe
263
264 (defun imprimer-anglais (etape2)
265   (mapcar #'caaar etape2) )
266
267 ; NAME : imprimer-cat ; renvoie que les cat (on utilise etape2 car
268 ; à partir de etape3 on a modifié la construction du dico > en vue des comparaisons)
269 ; ARGS : liste
270 ; USAGES : (imprimer-cat etape2) (imprimer-cat etape2) ("{adv}" "{proverb}" "{prefix}" ...
271 ; GLOBALS : none
272 ; CALL :
273 ; USER : printe
274
275 (defun imprimer-cat (etape2)
276   (mapcar #'caadar etape2) )
277
278 ; NAME : imprimer-espagnol ; renvoie que les mots en esp
279 ; (on utilise etape2 car à partir de etape3
280 ; on a modifié la construction du dico > en vue des comparaisons de .txt)
281 ; ARGS : liste &aux resultat
282 ; USAGES : (imprimer-espagnol etape2) > ; (("hacia atrás") ("un día más un día menos")
283 ; ("hiper-" "-ico") ("majestad {f}") ("mandarín {m}", chino mandarín {m})) ("bah")
284 ; GLOBALS : none
285 ; CALL :
286 ; USER : printe
287
288 (defun imprimer-espagnol (etape2 &aux resultat)
289   (loop for element in etape2 do ; pour chaque élément de etape2
290     (push (nth 1 element) resultat) ) ; on ajoute le cdr à resultat
291   (reverse resultat) ) ; on renvoie la liste inversée
292
293
294
295 ; NAME : printe ; reconstruit la liste de départ à partir de étape 2
296 ; (car au delà on a modifié la construction du dico

```

```

297 ;           pour supprimer les virgules entre les trads)
298 ; ARGS : liste &aux parties-anglaises parties-cat parties-espagnoles result
299 ; USAGES : (printe etape2)> ("aback" "{adv}" "hacia atrás")
300 ; GLOBALS : none
301 ; CALL : imprimer-anglais, imprimer-cat, imprimer-espagnol
302 ; USER : main
303
304 (defun printe (liste &aux parties-anglaises parties-cat parties-espagnoles result)
305   (setq parties-anglaises (imprimer-anglais liste)) ; reconstruction des parties anglaises
306   (setq parties-cat (imprimer-cat liste)) ; reconstruction des catégories
307   (setq parties-espagnoles (imprimer-espagnol liste)) ; reconstruction des parties espagnoles
308   (setq result nil) ; liste vide pour les résultats
309   (loop while parties-anglaises do
310     (push (list (car parties-anglaises) (car parties-cat) (caar parties-espagnoles)) result)
311     ; on ajoute le mot anglais + sa cat + sa trad esp à résultat
312     (setq parties-anglaises (cdr parties-anglaises))
313     (setq parties-cat (cdr parties-cat))
314     (setq parties-espagnoles (cdr parties-espagnoles)) )
315   (reverse result) ) ; on inverse résultat
316
317
318
319
320 ; NAME : myprint ; la reconstruction du dico à partir de étape2
321 ; USAGES :  (("aback" "{adv}" "hacia atrás")...
322 ; USER :
323 ; (setq myprint (printe etape2))
324
325
326 ; NAME : imprimer ; imprimer la liste reconstruite dans le fichier
327 ; ARGS : liste (étape2) + fichier sortie
328 ; USAGES : (imprimer myprint "test.txt") > nil
329 ; GLOBALS : none
330 ; CALL : mon_format
331 ; USER : main
332
333 (defun imprimer (liste fichier)
334   (with-open-file (out fichier
335     :direction :output
336     :if-exists :supersede )
337     (mon_format liste out) ) )
338
339 ; NAME : mon_format ; formatage de la liste pour l'impression
340 ; ARGS : liste (étape2) + fichier sortie
341 ; USAGES : (mon_format myprint out) > nil
342 ; GLOBALS : none
343 ; CALL : none
344 ; USER : imprimer
345
346 (defun mon_format (liste fichier_sortie)
347   (if (atom liste) ; si fin de la liste
348     nil ; on renvoie nil
349     (loop for item in liste do ; sinon pour chaque elt de la L
350       (format fichier_sortie "~a ~a ~a~%"
351         (car item) ; mot anglais
352         (cadr item) ; cat
353         (caddr item) ) ) ) ; partie espagnole
354
355
356 ; NAME : acces-ligne ; parcourt le dictionnaire
357 ; ARGS : etape4 stream

```

```

358 ; USAGES : (acces-ligne etape4 acces-ligne.txt) > Accès à une ligne (parcours du dico):
359 ;      ((a . {art}) ((una . {art}) (uno . {art})))
360 ; GLOBALS : none
361 ; CALL : none
362 ; USER : test
363
364 (defun acces-ligne (etape4 stream)
365   (format stream "Accès à une ligne (parcours du dico):~%" )
366   (loop for ligne in etape4 do
367     (format stream "~a~%" ligne) ) )
368
369 ; NAME : acces-anglais (car de la ligne)
370 ; ARGS : etape4 stream
371 ; USAGES : (acces-anglais etape4 acces-anglais.txt) ; Accès à l'anglais (car de la ligne):
372 ;      (a . {art})
373 ; GLOBALS : none
374 ; CALL : none
375 ; USER : test
376
377 (defun acces-anglais (etape4 stream)
378   (format stream "Accès à l'anglais (car de la ligne):~%" )
379   (loop for ligne in etape4 do
380     (format stream "~a~%" (car ligne)) ) )
381
382 ; NAME : acces-espagnol (cadr de la ligne)
383 ; ARGS : etape4 stream
384 ; USAGES : (acces-espagnol etape4 stream) > Accès à l'espagnol (cadr de la ligne):
385 ;      ((una . {art}) (uno . {art}))
386 ; GLOBALS : none
387 ; CALL : none
388 ; USER : test
389
390 (defun acces-espagnol (etape4 stream)
391   (format stream "Accès à l'espagnol (cadr de la ligne):~%" )
392   (loop for ligne in etape4 do
393     (format stream "~a~%" (cadr ligne)) ) )
394
395 ; NAME : acces-mot-anglais (caar de la ligne)
396 ; ARGS : etape4 stream
397 ; USAGES : (acces-mot-anglais etape4 stream) > Accès au mot anglais (caar de la ligne): a
398 ; GLOBALS : none
399 ; CALL : none
400 ; USER : test
401
402 (defun acces-mot-anglais (etape4 stream)
403   (format stream "Accès au mot anglais (caar de la ligne):~%" )
404   (loop for ligne in etape4 do
405     (format stream "~a~%" (caar ligne)) ) )
406
407 ; NAME : acces-cat-anglais (cdar de la ligne)
408 ; ARGS : etape4 stream
409 ; USAGES : (acces-cat-anglais etape4 stream) > Accès à la catégorie de l'anglais
410 ;      (cdar de la ligne): {art}
411 ; GLOBALS : none
412 ; CALL : none
413 ; USER : test
414
415 (defun acces-cat-anglais (etape4 stream)
416   (format stream "Accès à la catégorie de l'anglais (cdar de la ligne):~%" )
417   (loop for ligne in etape4 do
418     (format stream "~a~%" (cdar ligne)) ) )

```

```

419
420
421 ; NAME : acces-traduction (parcours du cadr de la ligne)
422 ; ARGS : etape4 stream
423 ; USAGES : (acces-traduction etape4 stream) > Accès à une traduction
424 ;          (parcours du cadr de la ligne): (una . {art})
425 ; GLOBALS : none
426 ; CALL : none
427 ; USER : test
428
429 (defun acces-traduction (etape4 stream)
430   (format stream "Accès à une traduction (parcours du cadr de la ligne):~%" )
431   (loop for ligne in etape4 do
432     (loop for traduction in (cadr ligne) do
433       (format stream "~a~%" traduction) ) ) )
434
435 ; NAME : acces-mot-espagnol (parcours car de la sous-liste)
436 ; ARGS : etape4 stream
437 ; USAGES : (acces-mot-espagnol etape4 stream) > Accès à un mot espagnol
438 ;          (car de la sous-liste): una
439 ; GLOBALS : none
440 ; CALL : none
441 ; USER : test
442
443 (defun acces-mot-espagnol (etape4 stream)
444   (format stream "Accès à un mot espagnol (car de la sous-liste):~%" )
445   (loop for ligne in etape4 do
446     (loop for traduction in (cadr ligne) do
447       (format stream "~a~%" (car traduction)) ) ) )
448
449 ; NAME : acces-cat-trad (parcours cdr de la sous-liste)
450 ; ARGS : etape4
451 ; USAGES : (acces-cat-trad etape4 stream) > Accès à la catégorie d'une traduction
452 ;          (cdr de la sous-liste): {art}
453 ; GLOBALS : none
454 ; CALL : none
455 ; USER : test
456
457 (defun acces-cat-trad (etape4 stream)
458   (format stream "Accès à la catégorie d'une traduction (cdr de la sous-liste):~%" )
459   (loop for ligne in etape4 do
460     (loop for traduction in (cadr ligne) do
461       (format stream "~a~%" (cdr traduction)) ) ) )
462
463
464 ; NAME : test ; pour vérifier les accès aux données
465 ;          imprime chaque fonction dans un fichier
466 ; ARGS : etape4
467 ; USAGES : (test etape4) > nil
468 ; GLOBALS : none
469 ; CALL : acces-ligne, acces-anglais, acces-espagnol, acces-mot-anglais
470 ; acces-cat-anglais, acces-traduction, acces-mot-espagnol, acces-cat-trad
471 ; USER : main
472
473 (defun test (etape4)
474   (with-open-file
475     (stream "ligne.txt"
476       :direction :output
477       :if-exists :supersede
478       :if-does-not-exist :create )
479     (acces-ligne etape4 stream) )

```

```

480 (with-open-file
481   (stream "anglais.txt"
482     :direction :output
483     :if-exists :supersede
484     :if-does-not-exist :create )
485   (acces-anglais etape4 stream) )
486 (with-open-file
487   (stream "espagnol.txt"
488     :direction :output
489     :if-exists :supersede
490     :if-does-not-exist :create )
491   (acces-espagnol etape4 stream) )
492 (with-open-file
493   (stream "mot-anglais.txt"
494     :direction :output
495     :if-exists :supersede
496     :if-does-not-exist :create )
497   (acces-mot-anglais etape4 stream) )
498 (with-open-file
499   (stream "cat-anglais.txt"
500     :direction :output
501     :if-exists :supersede
502     :if-does-not-exist :create )
503   (acces-cat-anglais etape4 stream) )
504 (with-open-file
505   (stream "traduction.txt"
506     :direction :output
507     :if-exists :supersede
508     :if-does-not-exist :create )
509   (acces-traduction etape4 stream) )
510 (with-open-file
511   (stream "mot-espagnol.txt"
512     :direction :output
513     :if-exists :supersede
514     :if-does-not-exist :create )
515   (acces-mot-espagnol etape4 stream) )
516 (with-open-file
517   (stream "cat-trad.txt"
518     :direction :output
519     :if-exists :supersede
520     :if-does-not-exist :create )
521   (acces-cat-trad etape4 stream) ) )
522
523
524
525 ; NAME : main ; la fonction principale realise les traitements successifs
526 ; ARGS : entree sortie_reconstruction &aux etape1 etape2 etape3 etape4 myprint
527 ; USAGES : (main "mon_dico.txt" "reconstruction_dico.txt") > nil
528 ; GLOBALS :
529 ; CALL : lire-fichier, parentheses, defcat, structure_en_cours, printe, imprimer, test
530 ; USER : toplevel
531
532 (defun main (entree sortie_reconstruction &aux etape1 etape2 etape3 etape4 myprint)
533   (setq etape1 (lire-fichier entree)) ; lit le fichier ligne par ligne
534   (setq etape2 (parentheses etape1)) ; ajoute les parenthèses
535   (setq etape3 (structure_en_cours etape2)) ; isole chaque différente traduction possible
536   (setq etape4 (forme_finale etape3 etape3))
537   ; ajoute les catégories aux différentes traductions
538   (setq myprint (printe etape2))
539   ; reconstruit la liste de départ à partir de l'étape 2 (avant modification du contenu du dictionnaire)
540   (imprimer myprint sortie_reconstruction)

```

```

541 ; imprime le résultat de notre reconstruction en sortie
542 (test etape4) ) ; pour vérifier les accès aux données
543
544
545
546 ; NAME : appelle main pour réaliser les traitements sur fichier test et dico original
547 ; USAGES : vérification bon fonctionnement programme complet
548 ; USER :
549
550 (main "mon_dico.txt" "reconstruction_dico.txt")
551
552
553 ; NAME : traduction
554 ; ARGS : mot categorie &aux etape1 etape2 etape3 etape4 mot-trouve
555 ; USAGES : (traduction 'a '{art}) > una {art} uno {art}
556 ; GLOBALS : none
557 ; CALL : none
558 ; USER : toplevel
559
560 (defun traduction (mot categorie &aux etape1 etape2 etape3 etape4 mot-trouve)
561   (setq etape1 (lire-fichier "mon_dico.txt")) ; lit le fichier ligne par ligne
562   (setq etape2 (parentheses etape1)) ; ajoute les parenthèses
563   (setq etape3 (structure_en_cours etape2)) ; isole chaque différente trad
564   (setq etape4 (forme_finale etape3 etape3)) ; ajoute les cat aux différentes trad
565   (loop for ligne in etape4 do ; pour chaque ligne du dico
566     (when (and (string-equal (caar ligne) mot) (string-equal (cdar ligne) categorie))
567       ; si le mot et la catégorie correspondent
568       (setq mot-trouve t) ; on a trouvé le mot
569       (loop for traduction in (cadr ligne) do ; pour chaque traduction
570         (format t "~A ~A~%" (car traduction) (cdr traduction)) ) ) )
571       ; on imprime la traduction
572   (unless mot-trouve ; si le mot n'a pas été trouvé
573     (format t "Il n'y a pas de traduction disponible pour le mot ~A de catégorie ~A~%" mot categorie) )
574   (values) ) ; on renvoie rien (pour éviter le nil)

```

.2 Le code source "dictionnaire.c"

Voici l'intégralité du code source `dictionnaire`.

```
1  /* # Nom ..... : dictionnaire.c
2  # Rôle ..... : un dictionnaire bilingue eng-esp
3  # Auteur ..... : Avrile Floro
4  # Version ..... : V0.1 du 17/11/23
5  # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6  # Usage : pour le dico bilingue > make > puis ./dictionnaire
7  # Usage : pour les tests de reconstruction > make -f makefile_tests
8  #           puis > ./dictionnaire_test */
9
10
11 #include "sys.h"
12 #include "structure.h"
13
14
15 int main(int argc, char *argv[]) {
16
17     # ifdef TEST /* si -DTEST on initialise les fichiers */
18     initialisation_fichiers();
19     # endif
20
21     FILE* f_entree = NULL;
22     char p_chaine[1000];
23
24     /* ouverture du fichier en lecture */
25     f_entree = fopen("mon_dico.txt", "r");
26
27     list L = NULL ;
28
29     if (f_entree != NULL) {
30         /* mise en mémoire du dico */
31         while (fgets(p_chaine, sizeof(p_chaine), f_entree)) {
32             p_chaine[strcspn(p_chaine, "\n")] = '\0'; /* on supprime le \n */
33             L = cons(strdup(p_chaine), L); /* ajoute copie ligne à la liste */
34         }
35         fclose(f_entree);
36     }
37
38     /* inversion de la liste pour conserver l'ordre */
39     L = reverse(L);
40     /* obtention de notre dictionnaire traité */
41     une_entree dictionnaire = div_eng_esp(L);
42
43     #ifdef TEST /* si -DTEST on fait les tests */
44     creation_tests(dictionnaire);
45
46     #else /* sinon traducteur en ligne de commande */
47     while (true) {
48         traduction(dictionnaire);
49         printf("\n");
50     }
51     #endif
52     return 0;
53 }
54
55 /*; NAME : traduction ; donne l'esp d'un mot eng
56 ; ARGS : dictionnaire
```

```

57 ; USAGES : traduction(dictionnaire)
58 ; GLOBALS : none
59 ; CALL : none
60 ; USER : main */
61
62 void traduction(une_entree dictionnaire){
63
64     /* on récupère la ligne entière */
65     char ligne[100];
66     printf("Quel mot voulez-vous traduire ?\n");
67     fgets(ligne, sizeof(ligne), stdin);
68
69     /* traitement du mot */
70     string mot = strtok(ligne, "{"); /* séparation avant le { */
71     int len = strlen(mot);
72     mot[len-1]='\0'; /* on supprime l'espace */
73
74     /* traitement de la catégorie */
75     string cat = strtok(NULL, "");
76     if (cat == NULL){
77         printf("Usage: mot_anglais {catégorie_recherchée}\n");
78         return;
79     }
80
81     string categorie = malloc(strlen(cat) + 2); /*pour rajouter le { */
82     categorie[0]='\0'; /* initialisation */
83     strcat(categorie, "{"); /* rajout { */
84     strcat(categorie, cat); /* copie du strtok */
85     int len2 = strlen(categorie);
86     categorie[len2-1]='\0'; /* suppression dernier caractère*/
87
88
89     bool found = false;
90     une_entree p_dico = dictionnaire;
91     /* utilisation copie pour pas parcourir le dico original */
92
93     while (p_dico != NULL && found == false) { /* tant qu'il y a des mots */
94
95         if ((strcmp(p_dico->anglais->mot, mot) == 0) &&
96             (strcmp(p_dico->anglais->categorie->chaine, categorie) == 0)) {
97             /* si mot et catégorie sont identiques */
98
99
100             trad_espagnol temp = p_dico->traduction;
101             /* utilisation ptr temp pour conserver ordre original dico */
102
103             while (temp != NULL) { /* tant qu'il y a des trad du mot */
104                 printf("%s %s", temp->espagnol->mot, temp->espagnol->categorie->chaine);
105                 /* on les imprime */
106                 if (temp->cdr != NULL) { /* et si ce n'est pas le dernier mot */
107                     printf(", "); /* on ajoute une virgule entre les trads */
108                 }
109
110                 temp = temp->cdr; /* on parcourt les trads */
111             }
112
113             found = true; /* arrêt du parcours du dico */
114             printf("\n");
115         }
116         p_dico = p_dico->cdr; /* on parcourt les mots du dico */
117     }

```



```

118
119     if (!found){
120         printf("Aucune traduction n'a été trouvée. \n");
121     }
122 }
123
124
125 /*; NAME : cons ; ajout d'un élément au début d'une l
126 ; ARGS : elt, L
127 ; USAGES : cons(elt, L)
128 ; GLOBALS : none
129 ; CALL : none
130 ; USER : main */
131
132 list cons(void * elt, list L)
133 {
134     list Cons = malloc(sizeof(struct Doublet)) ;
135     Cons->car = elt;
136     Cons->cdr = L;
137
138     return Cons ;
139 }
140
141 /*; NAME : reverse ; inverse ordre liste
142 ; ARGS : L
143 ; USAGES : reverse(L)
144 ; GLOBALS : none
145 ; CALL : none
146 ; USER : main */
147
148 list reverse(list L) {
149     list R = NULL; // liste vide
150     while (L != NULL) {
151         list temp = cdr(L); // sauvegarde du cdr de L
152         cdr(L) = R; // le cdr de L pointe sur R
153         R = L; // R pointe sur L
154         L = temp; // L pointe sur le prochain élt de L
155     }
156     return R;
157 }
158
159
160 /*; NAME : div_eng_esp ; traitement données complet dico
161 ; ARGS : L
162 ; USAGES : div_eng_esp(L)
163 ; GLOBALS : none
164 ; CALL : eng_cat, spanish
165 ; USER : main */
166
167 une_entree div_eng_esp(list L)
168 {
169     une_entree debut_dico = NULL; /* pour connecter les cdr */
170     une_entree precedent = NULL;
171
172     while (L != NULL){ /* parcourt de la liste */
173
174         une_entree un_mot = malloc(sizeof(struct MotDictionnaire));
175         string ligne = (string)(car(L)); /* mise en mémoire des lignes */
176
177         /* division des catégories */
178         /* anglais */

```

```

179 string eng = strtok(ligne, "}"); /* premier token jusqu'à \} */
180 /* espagnol */
181 string esp = strtok(NULL, ""); /* 2e token jusqu'à la fin */
182
183 /* traitement de l'anglais */
184 string engl = malloc(strlen(eng) + 2); /* pour rajouter le } */
185 strcpy(engl, eng);
186 strcat(engl, "}"); /* rajout { manquant */
187
188 /* formation entrée anglais */
189 un_mot->anglais = eng_cat(engl);
190
191 /* formation entrée traduction */
192 un_mot->traduction = spanish(esp, un_mot->anglais->categorie->chaine);
193
194 /* connexion des entrées dico via le cdr*/
195 if (debut_dico == NULL) { /* si début de la liste */
196     debut_dico = un_mot; /* 1er élément */
197 }
198 else { /* si pas début de la liste */
199     precedent->cdr = un_mot; /* ajout à la fin de la liste */
200 }
201 precedent = un_mot; /* parcours des élts */
202 L = cdr (L);
203
204 }
205
206 return debut_dico; /* renvoie le dico sous la bonne forme */
207 }
208
209
210 /*; NAME : eng_cat ; traitement données anglais
211 ; ARGS : string english_part
212 ; USAGES : eng_cat(english_part)
213 ; GLOBALS : none
214 ; CALL : assoc_categorie
215 ; USER : div_eng_esp */
216
217 forme_anglais eng_cat(string english_part){
218
219     forme_anglais anglais = malloc(sizeof(struct Base));
220
221     string original_str = strdup(english_part);
222
223     /* traitement du mot anglais */
224     string mot_anglais = strtok(english_part, "{");
225     int len = strlen(mot_anglais); /* retire espace à la fin */
226     mot_anglais[len-1]='\0';
227
228     /* traitement de la catégorie anglaise */
229     string cat_ang = strtok(NULL, "");
230     /* travail sur les chaînes pour rajouter le { au début */
231     string debut = "{";
232     string cat_anglais = malloc(strlen(cat_ang)+2);
233     strcpy(cat_anglais, debut);
234     strcat(cat_anglais, cat_ang);
235
236     /* remplissage des éléments de struct [anglais] */
237     anglais->mot= strdup(mot_anglais);
238     anglais->original = strdup(original_str);
239     anglais->categorie=assoc_categorie(cat_anglais);

```

```

240
241     return anglais; /* renvoie la struct remplie */
242 }
243
244
245 /*; NAME : spanish ; traitement données traduction esp
246 ; ARGS : string esp_part, string cat
247 ; USAGES : spanish(esp_part, cat)
248 ; GLOBALS : none
249 ; CALL : subdiv_esp
250 ; USER : div_eng_esp */
251
252 trad_espagnol spanish(string esp_part, string cat){
253
254     trad_espagnol premiere_trad = NULL;
255     trad_espagnol trad_prec = NULL;
256
257     string token = NULL;
258
259     string copy_esp_part = strdup(esp_part);
260     char *ptr;
261     token = strtok_r(copy_esp_part, ",", &ptr);
262     /* strtok_r car subdiv_esp utilise aussi strtok */
263
264     while (token) {
265         trad_espagnol espagnol_trad = malloc(sizeof(struct Trad));
266         forme_espagnol traduction = subdiv_esp(token, cat);
267
268         espagnol_trad->espagnol= traduction;
269         espagnol_trad->cdr = NULL; /* initialisation à NULL pour éviter bugs! */
270
271         /* connexion des trads via le cdr */
272         if (premiere_trad == NULL){ /* si début de liste */
273             premiere_trad = espagnol_trad; /* 1er élément */
274         }
275         else { /* si pas début de la liste */
276             trad_prec->cdr=espagnol_trad; /* ajout comme dernier élt */
277         }
278         trad_prec = espagnol_trad; /* parcours des élt */
279         token= strtok_r(NULL, ",", &ptr); /* token suivant */
280     }
281     return premiere_trad; /* retour à partir du 1er élt */
282 }
283
284
285 /*; NAME : subdiv_esp ; traitement données forme esp
286 ; ARGS : string esp_original, string cat
287 ; USAGES : subdiv_esp(esp_original, cat)
288 ; GLOBALS : none
289 ; CALL : assoc_categorie
290 ; USER : spanish */
291
292 forme_espagnol subdiv_esp(string esp_original, string cat){
293
294     forme_espagnol espagnol = malloc(sizeof(struct Base));
295     string str_original = strdup(esp_original);
296     string esp_original_copy = strdup(str_original);
297
298     /* traitement du mot espagnol */
299     string tok = strtok(esp_original_copy, "{");
300     int len = strlen(tok);

```

```

301 tok[len]='\0'; /*suppression espace final */
302 string tok_bis = &tok[1]; /* suppression espace avant */
303
304 /* catégorie esp est le strktok suivant */
305 string tok2 = strtok(NULL, "");
306
307 if (tok2 == NULL){ /* s'il n'y a pas de catégorie esp */
308     espagnol->mot = tok_bis; /* le mot est le mot traité */
309     espagnol->original = strdup(str_original); /* trace de l'original */
310     espagnol->categorie = assoc_categorie(cat);
311     /* on utilise la catégorie du mot anglais */
312 }
313
314 else { /* s'il y a une catégorie du mot espagnol fournie */
315     /* traitement de la catégorie */
316     string cut_cat = tok2;
317     int len_cut = strlen(tok_bis);
318     tok_bis[len_cut-1]='\0'; /* supprimer dernier caractère */
319     string ma_cat = (malloc(strlen(cut_cat)+2));
320     ma_cat[0] = '\0'; /* initialisation */
321     strcat(ma_cat, "{"); /* ajout du { */
322     strcat(ma_cat, cut_cat); /* obtention cat complete */
323
324     /* on effectue les associations pour espagnol */
325     espagnol->mot = tok_bis;
326     espagnol->original = strdup(str_original);
327     espagnol->categorie = assoc_categorie(ma_cat);
328 }
329 return espagnol; /* on renvoie la struct formée */
330 }

```

.3 L'intégralité du fichier "category.c"

Voici l'intégralité du fichier `category.c`.

```
1 /* # Nom ..... : category.c
2 # Rôle ..... : mäj des cat grammaticales des mots
3 # Auteur ..... : Avrile Floro
4 # Version ..... : V0.1 du 17/11/23
5 # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 # Usage : compilé via "make" ou "make -f makefile_tests" */
7
8 #include "sys.h"
9 #include "structure.h"
10
11 #define NOUN 1
12 #define VERB 2
13 #define ADJECTIVE 3
14 #define ADVERB 4
15 #define INTERJECTION 5
16 #define PRONOUN 6
17 #define PREFIX 7
18 #define SUFFIX 8
19 #define PROVERB 9
20 #define PROPER_NOUN 10
21 #define PHRASE 11
22 #define PREP_PHRASE 12
23 #define CONJUNCTION 13
24 #define DETERMINER 14
25 #define CONTRACTION 15
26 #define ART 16
27 #define SYMBOL 17
28 #define CARDINAL_NUM 18
29 #define ACRONYM 19
30 #define PREPOSITION 20
31 #define ABBREVIATION 21
32 #define INITIALISM 22
33 #define PARTICLE 23
34 #define NUM 24
35 #define NOM_MASCULIN 25
36 #define NOM_FEMININ 26
37 #define NOM_PLURIEL 27
38 #define NOM_MASCULIN_PLURIEL 28
39 #define NOM_FEMININ_PLURIEL 29
40 #define NOM_EPICENE 30
41 #define SINGULIER 31
42
43
44 /*; NAME : assoc_categorie ; associe str cat et int du mot
45 ; ARGS : string cat_originale
46 ; USAGES : assoc_categorie(cat_originale)
47 ; GLOBALS : none
48 ; CALL : none
49 ; USER : eng_cat, subdiv_esp */
50
51 la_categorie assoc_categorie(string cat_originale){
52
53     la_categorie cat = (malloc(sizeof(struct Category)));
54     string cat_dup = strdup(cat_originale);
55     cat->chaîne = cat_dup; /* association cat->chaîne*/
56     /* définition du int de la catégorie */
```

```

57 if (strcmp(cat_dup, "{n}") == 0 || strcmp(cat_dup, "{n} ") == 0) {
58     cat->cat = NOUN;
59 }
60 else if (strcmp(cat_dup, "{v}") == 0) {
61     cat->cat = VERB;
62 }
63 else if (strcmp(cat_dup, "{adj}") == 0) {
64     cat->cat = ADJECTIVE;
65 }
66 else if (strcmp(cat_dup, "{adv}") == 0) {
67     cat->cat = ADVERB;
68 }
69 else if (strcmp(cat_dup, "{interj}") == 0) {
70     cat->cat = INTERJECTION;
71 }
72 else if (strcmp(cat_dup, "{pron}") == 0) {
73     cat->cat = PRONOUN;
74 }
75 else if (strcmp(cat_dup, "{prefix}") == 0) {
76     cat->cat = PREFIX;
77 }
78 else if (strcmp(cat_dup, "{suffix}") == 0) {
79     cat->cat = SUFFIX;
80 }
81 else if (strcmp(cat_dup, "{proverb}") == 0) {
82     cat->cat = PROVERB;
83 }
84 else if (strcmp(cat_dup, "{prop}") == 0) {
85     cat->cat = PROPER_NOUN;
86 }
87 else if (strcmp(cat_dup, "{phrase}") == 0) {
88     cat->cat = PHRASE;
89 }
90 else if (strcmp(cat_dup, "{prep phrase}") == 0) {
91     cat->cat = PRÉP_PHRASE;
92 }
93 else if (strcmp(cat_dup, "{conj}") == 0) {
94     cat->cat = CONJUNCTION;
95 }
96 else if (strcmp(cat_dup, "{determiner}") == 0) {
97     cat->cat = DETERMINER;
98 }
99 else if (strcmp(cat_dup, "{contraction}") == 0) {
100     cat->cat = CONTRACTION;
101 }
102 else if (strcmp(cat_dup, "{art}") == 0) {
103     cat->cat = ART;
104 }
105 else if (strcmp(cat_dup, "{symbol}") == 0) {
106     cat->cat = SYMBOL;
107 }
108 else if (strcmp(cat_dup, "{cardinal num}") == 0) {
109     cat->cat = CARDINAL_NUM;
110 }
111 else if (strcmp(cat_dup, "{acronym}") == 0) {
112     cat->cat = ACRONYM;
113 }
114 else if (strcmp(cat_dup, "{prep}") == 0) {
115     cat->cat = PREPOSITION;
116 }
117 else if (strcmp(cat_dup, "{abbr}") == 0) {

```

```

118     cat->cat = ABBREVIATION;
119 }
120 else if (strcmp(cat_dup, "{initialism}") == 0) {
121     cat->cat = INITIALISM;
122 }
123 else if (strcmp(cat_dup, "{particle}") == 0) {
124     cat->cat = PARTICLE;
125 }
126 else if (strcmp(cat_dup, "{num}") == 0) {
127     cat->cat = NUM;
128 }
129 else if (strcmp(cat_dup, "{m}") == 0) {
130     cat->cat = NOM_MASCULIN;
131 }
132 else if (strcmp(cat_dup, "{f}") == 0) {
133     cat->cat = NOM_FEMININ;
134 }
135 else if (strcmp(cat_dup, "{p}") == 0) {
136     cat->cat = NOM_PLURIEL;
137 }
138 else if (strcmp(cat_dup, "{m.p}") == 0) {
139     cat->cat = NOM_MASCULIN_PLURIEL;
140 }
141 else if (strcmp(cat_dup, "{f.p}") == 0) {
142     cat->cat = NOM_FEMININ_PLURIEL;
143 }
144 else if (strcmp(cat_dup, "{f}{m}") == 0 || strcmp(cat_dup, "{f}/{m}") == 0 ||
145 strcmp(cat_dup, "{m}{f}") == 0 || strcmp(cat_dup, "{m}/{f}") == 0){
146     cat->cat = NOM_EPICENE;
147 }
148 else if (strcmp(cat_dup, "{s}") == 0) {
149     cat->cat = SINGULIER;
150 }
151
152
153 return cat; /* renvoie la catégorie */
154 }

```

.4 L'intégralité du fichier "test.c"

Voici l'intégralité du fichier `test.c`.

```
1  /* # Nom ..... : test.c
2  # Rôle ..... : initialise les fichiers et lance les tests
3  # Auteur ..... : Avrië Floro
4  # Version ..... : V0.1 du 17/11/23
5  # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6  # Usage : compilé seulement pour réaliser les tests
7  # Usage : pour les tests de reconstruction > make -f makefile_tests
8  #      puis > ./dictionnaire_test */
9
10
11 #include "sys.h"
12 #include "structure.h"
13
14
15 /*; NAME : initialisation_fichier(); init fichier en W si -DTEST
16 ; ARGS :
17 ; USAGES : initialisation_fichier()
18 ; GLOBALS : none
19 ; CALL : none
20 ; USER : main */
21
22 void initialisation_fichiers(){
23     /* initialisation des fichiers en W pour effacer les contenus précédents */
24     /* fermeture */
25     FILE *f_mot_anglais = fopen("mot_anglais.txt", "w");
26     fclose(f_mot_anglais);
27     FILE *f_all_words_traduc_esp = fopen("all_words_traduc_esp.txt", "w");
28     fclose(f_all_words_traduc_esp);
29     FILE *f_all_trad_esp_treated = fopen("all_trad_esp_treated.txt", "w");
30     fclose(f_all_trad_esp_treated);
31     FILE *f_cat_anglais = fopen("cat_mot_anglais.txt", "w");
32     fclose(f_cat_anglais);
33     FILE *f_cat_esp = fopen("cat_mot_esp.txt", "w");
34     fclose(f_cat_esp);
35     FILE *f_reconstruction = fopen("reconstruction.txt", "w");
36     fclose(f_reconstruction);
37     FILE *f_anglais = fopen("tout_anglais.txt", "w");
38     fclose(f_anglais);
39     FILE *f_espagnol = fopen("trad_espagnol.txt", "w");
40     fclose(f_espagnol);
41 }
42
43
44 /*; NAME : creation_tests ; lance les tets vers fichiers en A
45 ; ARGS : dictionnaire
46 ; USAGES : creation_tests(dictionnaire)
47 ; GLOBALS : none
48 ; CALL : none
49 ; USER : main */
50
51 void creation_tests(une_entree dictionnaire){
52
53     une_entree p_dico = dictionnaire; /* ptr pour pas modif original */
54
55     while (p_dico != NULL){
56         /* ouverture fichiers en mode append pour ajouter à chaque boucle */
```



```

57  /* fermeture des fichiers */
58  FILE *f_mot_anglais = fopen("mot_anglais.txt", "a");
59  fprintf(f_mot_anglais, "%s\n", p_dico->anglais->mot);
60  fclose(f_mot_anglais);
61  FILE *f_cat_anglais = fopen("cat_mot_anglais.txt", "a");
62  fprintf(f_cat_anglais, "%s\n", p_dico->anglais->categorie->chaine);
63  fclose(f_cat_anglais);
64  FILE *f_anglais = fopen("tout_anglais.txt", "a");
65  fprintf(f_anglais, "%s\n", p_dico->anglais->original);
66  fclose(f_anglais);
67  FILE *f_reconstruction = fopen("reconstruction.txt", "a");
68  fprintf(f_reconstruction, "%s", p_dico->anglais->original);
69  fclose(f_reconstruction);
70
71  trad_espagnol temp = p_dico->traduction; /* ptr pour pas altérer original */
72  while(temp != NULL){ /* parcourt des trads */
73
74      /* ouverture en mode append */
75      FILE *f_all_words_traduc_esp = fopen("all_words_traduc_esp.txt", "a");
76      FILE *f_all_trad_esp_treated = fopen("all_trad_esp_treated.txt", "a");
77      FILE *f_cat_esp = fopen("cat_mot_esp.txt", "a");
78      FILE *f_espagnol = fopen("trad_espagnol.txt", "a");
79      f_reconstruction = fopen("reconstruction.txt", "a");
80
81      /* impression dans les fichiers */
82      fprintf(f_all_words_traduc_esp, "%s\n", temp->espagnol->mot);
83      fprintf(f_all_trad_esp_treated, "%s %s\n", temp->espagnol->mot,
temp->espagnol->categorie->chaine);
84      fprintf(f_cat_esp, "%s\n", temp->espagnol->categorie->chaine);
85      fprintf(f_reconstruction, "%s", temp->espagnol->original);
86      fprintf(f_espagnol, "%s", temp->espagnol->original);
87
88      /* fermeture des fichiers */
89      fclose(f_all_words_traduc_esp);
90      fclose(f_all_trad_esp_treated);
91      fclose(f_cat_esp);
92      fclose(f_reconstruction);
93      fclose(f_espagnol);
94
95      if (temp->cdr != NULL){
96          f_reconstruction = fopen("reconstruction.txt", "a");
97          f_espagnol = fopen("trad_espagnol.txt", "a");
98          fprintf(f_reconstruction, ",");
99          fprintf(f_espagnol, ",");
100         fclose(f_reconstruction);
101         fclose(f_espagnol);
102     }
103
104     temp = temp->cdr; /* parcourt des trads du dico */
105 }
106 if (p_dico->cdr != NULL){
107     f_reconstruction = fopen("reconstruction.txt", "a");
108     FILE *f_espagnol = fopen("trad_espagnol.txt", "a");
109     fprintf(f_reconstruction, "\n");
110     fprintf(f_espagnol, "\n");
111     fclose(f_reconstruction);
112     fclose(f_espagnol);
113 }
114 p_dico = p_dico->cdr; /* parcourt des entrées du dico */
115 }
116 }

```

.5 L'intégralité du fichier "structure.h"

Voici l'intégralité du fichier `structure.h`.

```
1 /* # Nom ..... : structure.h
2 # Rôle ..... : proto des fonctions et déclaration des structures
3 # Auteur ..... : AvriLe Floro
4 # Version ..... : V0.1 du 17/11/23
5 # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 # Usage : fichier header */
7
8 typedef char * string ;
9 #define NULLCHAR "\0"
10
11 #define car(douplet) ((douplet)->car)
12 // macro plutôt que fonction, pour écrire car(x) = ...
13 #define cdr(douplet) ((douplet)->cdr)
14 // macro plutôt que fonction, pour écrire cdr(x) = ...
15
16
17 typedef struct Doublet {
18     void * car ;
19     struct Doublet * cdr ;
20 } * list ; // structure de Base ; list est un pointeur sur un doublet
21
22 typedef struct MotDictionnaire {
23     struct Base * anglais;
24     struct Trad * traduction;
25     struct MotDictionnaire * cdr ;
26 } * une_entree ;
27
28 typedef struct Base {
29     char* mot;
30     char* original;
31     struct Category * categorie;
32 } * forme_anglais, * forme_espagnol ;
33
34 typedef struct Trad {
35     struct Base * espagnol;
36     struct Trad * cdr;
37 } * trad_espagnol, * trad_anglais ;
38
39 typedef struct Category{
40     char* chaine;
41     int cat;
42 } * la_categorie ;
43
44
45 // les prototypes des fonctions
46
47 list cons(void *, const list) ; /* construction liste */
48
49 list reverse(list L) ; /* inversion liste */
50
51 une_entree div_eng_esp(list L); /* renvoie l'entrée */
52
53 forme_anglais eng_cat(string anglais); /* renvoie la forme anglaise */
54
55 trad_espagnol spanish(string esp, string cat); /* renvoie la trad esp */
56
```

```
57 forme_espagnol subdiv_esp(string token, string cat); /* renvoie la forme esp */
58
59 la_categorie assoc_categorie(string cat_originale); /* association catégorie */
60
61 void initialisation_fichiers(); /* initialisation préalable des fichiers en W */
62
63 void creation_tests(une_entree dictionnaire); /* création des tests si -DTEST */
64
65 void traduction(une_entree dictionnaire); /* programme de la traduction */
```

.6 L'intégralité du fichier "sys.h"

Voici l'intégralité du fichier `sys.h`.

```
1 #include <assert.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <unistd.h>
6 #include <sys/stat.h>
7 #include <sys/types.h>
8 #include <fcntl.h>
9 #include <sys/wait.h>
10 #include <stdbool.h>
11 #include <errno.h>
12 #include <time.h>
13 #include <signal.h>
14 # include <ctype.h>
```

.7 L'intégralité du makefile

Voici l'intégralité du fichier `makefile`.

```
1 # Nom ..... : makefile
2 # Rôle ..... : compile dictionnaire
3 # Auteur ..... : Avrile Floro
4 # Version ..... : V0.1 du 17/11/23
5 # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 # Usage : make
7
8
9 all: compile
10
11 compile:
12     gcc -Wall dictionnaire.c category.c -o dictionnaire
13
14 clean:
15     rm -f dictionnaire
```

.8 L'intégralité du makefile_tests

Voici l'intégralité du fichier `makefile_tests`.

```
1 # Nom ..... : makefile_tests
2 # Rôle ..... : compile dictionnaire_test
3 # Auteur ..... : AvriLe Floro
4 # Version ..... : V0.1 du 17/11/23
5 # Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 # Usage : make -f makefile_tests
7
8 all: compile
9
10 compile:
11     gcc -g -DTEST -Wall dictionnaire.c test.c category.c -o dictionnaire_test
12
13 clean:
14     rm -f dictionnaire_test
15     rm -f all_trad_esp_treated.txt
16     rm -f all_words_traduc_esp.txt
17     rm -f cat_mot_anglais.txt
18     rm -f cat_mot_esp.txt
19     rm -f mot_anglais.txt
20     rm -f reconstruction.txt
21     rm -f tout_anglais.txt
22     rm -f trad_espagnol.txt
```