TP n°7

Entrée/Sortie, Tableaux

Dans les exercices qui suivent, vous pourrez utiliser les fonctions prédéfinies suivantes pour réaliser des entrée/sortie avec des fichiers :

- `open_out : string -> out_channel` ouvre un fichier en sortie ;
- `close_out : out_channel -> unit` ferme un canal ;
- `output_string : out_channel -> string -> unit` écrit une chaîne de caractères sur un canal ;
- `open_in : string -> in_channel` ouvre un fichier en entrée ;
- `input_line : in_channel -> string` lit une ligne d'un canal d'entrée ou lève l'exception `End_of_file`.

Les fonctions suivantes seront également utiles pour manipuler des chaînes de caractères ou des caractères :

- `String.uppercase : string -> string` retourne la chaîne de caractères en entrée en majuscule ;
- `String.length : string -> int` donne la longueur d'une chaîne ;
- `String.sub : string -> int -> int -> string` retourne, pour `s n l` donnés, la sous-chaîne de `s` de longueur `l` commençant à la position `n` (la première position est `0`) ;
- `String.get : string -> int -> char` retourne le caractère en position `n` dans une chaîne de caractère `s`.
- `Char.code : char -> int` retourne le code ASCII d'un caractère.

Exercice 1. Écrire un programme OCAML qui produit un fichier de mots triés (un par ligne) à partir d'un fichier de mots donné (par exemple le fichier `mots` sur DIDEL). L'exécution du programme se fera avec la commande `ocaml tp7a.ml fichier1 fichier2` où `fichier1` est le nom du fichier en entrée et `fichier2` le nom du fichier produit (pour cela utiliser le module `Sys`).

Correcteur

Le but de cette partie est de programmer un correcteur d'orthographe rudimentaire qui étant donné un mot tapé par l'utilisateur vérifie d'abord s'il est correct et dans le cas contraire propose une liste de mots similaires. Pour simplifier le problème nous allons considérer des mots en anglais sans accents et apostrophes. Pour obtenir des mots similaires à un mot donné nous utilisons le code *soundex* (voir fr.wikipedia.org/wiki/Soundex) légèrement simplifié. L'idée est d'assigner à chaque mot un code et de regrouper tous les mots avec le même code dans une liste. Nous utiliserons des tableaux pour stocker les mots.

Le code soundex simplifié d'un mot est défini comme suit :

- On met le mot en majuscule
- On garde la première lettre
- On supprime toutes les occurrences des lettres : A, E, H, I, O, U, W, Y (à moins que ce ne soit la première lettre du mot)
- On attribue une valeur numérique aux lettres restantes de la manière suivante :
  - 1 = B, P
  - 2 = C, K, Q
  - 3 = D, T
  - 4 = L
  - 5 = M, N
  - 6 = R

  - 7 = G, J
  - 8 = X, Z, S
  - 9 = F, V
- Si deux lettres (ou plus) avec le même nombre sont adjacentes dans le mot *d'origine*, alors on ne retient que la première de ces lettres. Cette règle s'applique aussi à la première lettre.
- On renvoie le premier caractère ainsi que les trois premières valeurs numériques complétées par des zéros, si nécéssaire.

Par exemple, le code soundex de "hello" est `('H',4,0,0)`, celui de "Pagani" et de "ppigggeon" est `('P',7,5,0)` et celui de "programmer" est `('P',6,7,6)`.

Exercice 2. Définir un type `soundex`.

Exercice 3. Écrire une fonction `code_lettre : char -> int` qui étant donné une lettre majuscule donne son code numérique selon les règles du soundex (`-1` pour les voyelles et H et W).

Exercice 4. Écrire une fonction `rem_double : 'a list -> 'a list` qui à partir d'une liste l produit une liste avec tous les éléments de l dans le même ordre sans ceux dont le prédécesseur est le même. Par exemple pour `[1;1;2;2;2;-1;3;3]` on obtient `[1;2;-1;3]`.

Exercice 5. Écrire une fonction `string_to_list : string -> char list` qui permet à partir d'une chaîne de caractères d'obtenir la liste de ses caractères.

Exercice 6. Écrire une fonction `code_soundex : string -> soundex` qui à partir d'un mot renvoie son code soundex.

Pour stocker les codes soundex de plusieurs (beaucoup de mots) nous utilisons un tableau de 4 dimension qui contient une liste de mots qui ont le code soundex correspondant aux indices. Chaque lettre entre `A` et `Z` correspond à un entier entre 0 et 25. Donc, par exemple dans un tableau soundex `a` nous pouvons stocker à l'endroit `a.(15).(7).(5).(0)` la liste `["Pagani","pigeon",...]`. On définit un type soundexarray comme suit :

```
type soundexarray = string list array array array array;;
```

Exercice 7. Définir `arraysoundex : soundexarray` qui est un tableau soundex initialisé contenant des listes vides partout.

Exercice 8. Écrire une fonction `lirelignes : soundexarray -> string -> unit` qui remplit un tableau soundex avec les mots du fichier du nom donné. Tester votre fonction avec le fichier `words` sur DIDEL.

Exercice 9. Écrire une fonction `connu : soundexarray -> string -> string list` qui étant donné un tableau soundex et un mot déclenche l'exception `Correct` si le mot est stocké dans le tableau, sinon elle renvoie la liste des mots avec le même code ou alors `[]` s'il y en a pas.

Exercice 10. Écrire une fonction `testmots` qui prend comme entrée un tableau soundex et qui demande à l'utilisateur successivement d'entrer des mots. Si un mot est correct, on passe au mot suivant, sinon une liste de suggestions de mots similaires (avec le même code soundex) est affiché. L'utilisateur peut terminer en tapant "QUIT".

Exercice 11. ($\star\star$) Faire en sorte que tout marche aussi avec des mots contenant des accents et des apostrophes comme dans le fichier `mots`. Indication : le fichier `mots` est en format unicode.