TD 10 - Chaînes de caractères et dictionnaires

1 Les incontournables – chaînes

Exercice 1 Écrire une fonction `find_chr` qui reçoit une chaine de caractères *txt* et un caractère *need* et retourne l'indice de la première occurrence de *need* dans *txt* le cas échéant et -1 sinon.

Exercice 2 Écrire une fonction `palindrome : string -> bool` : qui indique si une chaîne de caractères est un palindrome.

2 Les incontournables – dictionnaires

Exercice 3 On choisit d'implémenter un dictionnaire à l'aide d'une liste de couples (clé, valeur).

1. Déclarer ce type polymorphe
2. Écrire les primitives sur cette structure :

   a. créer
   b. rechercher
   c. insérer
   d. cle_presente
   e. supprimer

3 Pour s'entrainer

Exercice 4 Écrire une fonction `voyelles` qui reçoit une chaîne de caractères en minuscule et renvoie la même chaine où toutes les voyelles ont été écrites en majuscule. On pourra utiliser la fonction `Char.escaped : char -> string` qui transforme un caractère en chaîne de caractères.

Exercice 5 En utilisant le module `Hashtbl`, on cherche à obtenir ce dictionnaire qui indique les valeurs des lettres au Scrabble :

```
{"A":1,"B":3,"C":3,"D":2,"E":1,"F":4,"G":2,"H":4,"I":1,"J":8,"K":10,"L":1...
```

à partir de la liste :

```
["AEILNORSTU", "DGM", "BCP", "FHV", "", "", "", "JQ", "", "KWXYZ"]
```

1. Écrire la fonction `list2dico`
2. Écrire la fonction `score : string -> int` qui donne le nombre de points pour un mot donné.

Correction 1 Soit on utilise une boucle `for` :

```
let find_chr txt need =
        let position = ref (-1) in
                for i = 0 to String.length txt - 1 do
                        if txt.[i] = need then position := i
                done;
                !position
;;
```

Mais cette solution donne la position de la dernière occurence contrairement à ce qui est demandé, il faut donc faire la boucle dans le sens inverse si on veut conserver cette idée en utilisant un `downto` :

```
let find_chr txt need =
        let position = ref (-1) in
                for i = String.length txt - 1 downto 0 do
                        if txt.[i] = need then position := i
                done;
                !position
;;
```

On peut améliorer si on parcourt la chaine jusqu'à trouver le caractère ou jusqu'à la fin si on ne le trouve pas. La double condition du while permet de sortir de la boucle dès que le caractère est trouvé :

```
let find_chr txt need =
  let ret = ref (-1) and i = ref 0  in
    while !i < String.length txt && !ret = -1 do
      if txt.[!i] = need then ret := !i;
      i := !i + 1;
    done;
  !ret;;
```

Enfin une version récursive est aussi intéressante, car elle permet aussi de sortir de la boucle dès que le caractère est trouvé :

```
let find_chr txt need =
    let rec regarde = function
        | i when i = String.length txt -> -1 (* pas trouvé definitivement *)
        | i when txt.[i] = need -> i            (* trouvé *)
        | i -> regarde (i+1)    (* on regarde le caractère suivant *)
    in regarde 0
;;
```

▷ Remarque : il existe déjà, dans le module `String`, une fonction `index` s c qui retourne la position de la première occurrence du caractère c dans la chaine s et signale l'erreur "Not_found" si celui-ci n'apparait pas, mais nous n'allons pas l'utiliser ici.

Correction 2 Une première solution où l'on parcourt tout le mot :

```
let palindrome mot =
  let nb = ref 0 and l = String.length mot - 1 in
    for i = 0 to l / 2 do
      if mot.[i] <> mot.[l-i] then nb := !nb + 1;
    done;
    !nb = 0;;
```

Une seconde où l'on s'arrête dès que 2 lettres diffèrent :

```
let palindrome mot =
  let nb = ref true and l = String.length mot - 1 and i = ref 0 in
    while (!i <= l / 2) && !nb do
      if mot.[!i] <> mot.[l - !i] then nb := false;
      i := !i + 1;
    done;
    !nb;;
```

Une dernière récursive :

```
let palindrome txt =
        let rec regarde i =
                let j = String.length txt -1 -i in match i with
                        | i when j <= i -> true
                        | i when txt.[i] <> txt.[j] -> false
                        | i -> regarde (i+1)
        in regarde 0
;;
```

Ou encore en regroupant les 2 derniers cas :

```
let palindrome txt =
        let rec regarde i =
                let j = String.length txt -1 -i in
                        if j <= i
                        then true
                        else (txt.[i] = txt.[j]) && (regarde (i+1))
        in regarde 0
;;
```

Correction 3 1. le type, qui est simplement une liste de couples :

```
type ('cle, 'valeur) dict = ('cle * 'valeur) list;;
```

2. Les primitives :

   a. Créer :

```
let creer () = [];;
```

   Noter que cette fonction polymorphe de type `creer unit -> 'a list`, on peut forcer la sortie si on le souhaite :

```
let creer () = ([]:('cle, 'valeur) dict);;
```

   b. Rechercher (pour la suite, on laisse les fonctions polymorphes) :

```
let rec rechercher d k =
        match d with
                  [] -> failwith "Non trouvée"
                | (k',v)::_ when k' = k -> v
                | _::q -> rechercher q k
;;

let dico = [("Maths",12) ; ("Français",13) ; ("Info",15)]:('cle, 'valeur)

# rechercher dico "Info";;
- : int = 15
# rechercher dico "Anglais";;
Exception: Failure "Non trouvée".
```

c. Ajouter : Une première solution de fénéant, mais qui fonctionne puisque la fonction `rechercher` renvoie la première occurrence trouvée :

```
let inserer d k v = (k,v)::d
```

Cependant cette solution n'est pas très satisfaisante dans la mesure où elle fait grandir la liste inutilement lors que l'on modifie la valeur d'une clé existante. Voici une solution plus optimisée :

```
let rec inserer d k v = match d with
          | [] -> [(k,v)]
          | (k',v')::t when k'=k -> (k,v)::t
          | h::t -> h::(inserer t k v)
;;
```

d. Pas de difficulté pour cette fonction :

```
let rec cle_presente d k = match d with
         | [] -> false
    | (k',v')::t when k'<>k -> cle_presente t k
         | _ -> true
;;
```

e. Suppression :

```
let rec supprimer d k = match d with
         | [] -> failwith "clé␣non␣présente"
    | (k',v')::t when k'=k -> t
         | h::t -> h::(supprimer t k)
;;
```

Correction 4 Une première solution récursive (car on ne peut pas modifier un caractère d'une chaine :

```
let voyelle txt =
        let rec voyelleR txt i =
                if i = String.length txt
                then ""
                else match txt.[i] with
                        | 'a' -> "A" ^ voyelleR txt (i+1)
                        | 'e' -> "E" ^ voyelleR txt (i+1)
                        | 'i' -> "I" ^ voyelleR txt (i+1)
                        | 'o' -> "O" ^ voyelleR txt (i+1)
                        | 'u' -> "U" ^ voyelleR txt (i+1)
                        | 'y' -> "Y" ^ voyelleR txt (i+1)
                        | _ -> Char.escaped (txt.[i]) ^ voyelleR txt (i+1)
        in voyelleR txt 0
;;
```

Une autre solution sans fonction auxiliaire :

```
let rec voyelle txt =
  if txt = ""
  then ""
  else match txt.[0] with
      | 'a' -> "A" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | 'e' -> "E" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | 'i' -> "I" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | 'o' -> "O" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | 'u' -> "U" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | 'y' -> "Y" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | _ -> String.make 1 (txt.[0]) ^ voyelle (String.sub txt 1 (String.length tx
;;
```

Voici deuxième solution utilisant une conversion de type `String -> Bytes`.

```
let voyelle txt =
        let b_txt = Bytes.of_string txt in
                for i = 0 to Bytes.length(b_txt) -1 do
        let c = Bytes.get b_txt i in
                if (c='a') || (c='e') ||(c='i') ||(c='o') ||(c='u') then
                Bytes.set b_txt i (char_of_int(int_of_char(c)-32));
                done;
                Bytes.to_string b_txt;;
```

Pour tester si `c` est une voyelle, on peut utiliser la fonction `contains` du module `String`. De même pour passer en majuscule, on peut utiliser la fonction `uppercase_ascii` du module `Char` :

```
let voyelle txt =
        let b_txt = Bytes.of_string txt in
                for i = 0 to Bytes.length(b_txt) -1 do
        let c = Bytes.get b_txt i in
                if String.contains "aeiuoy" c
                then Bytes.set b_txt i (Char.uppercase_ascii c);
                done;
                Bytes.to_string b_txt;;
```

Enfin, on peut se passer de la conversion de type en :

- Créant une fonction (`voyelle_maj : char -> char`) qui retourne le caractère donné en majuscule si c'est une voyelle et en minuscule sinon.
- Appliquant cette fonction à tous les caractères de la chaîne à l'aide de la fonction `map` du module `String`

```
let voyelles txt =
        let voyelle_maj c =
                if String.contains "aeiouy" c
                then Char.uppercase_ascii c else c
        in String.map voyelle_maj txt;;
```