TP 6 Programmation fonctionnelle

1 Fichiers CSV

On travaille sur des fichiers au format CSV (Comma-Separated Values). Un fichier CSV est un fichier texte dont le contenu représente un tableau. Chaque ligne du fichier correspond à une ligne du tableau. Les différentes valeurs composant une ligne sont séparées par des virgules. Par exemple

```
Identifiant,Partiel,Projet,Final
,0.3,0.3,0.4
ab12345678,12,15,14
cd98765432,10,,9
ef24681357,11,13
```

correspond au tableau

```
-----------------------------------------
| Identifiant | Partiel | Projet | Final |
-----------------------------------------
|             | 0.3     | 0.3    | 0.4   |
-----------------------------------------
| ab12345678  | 12      | 15     | 14    |
-----------------------------------------
| cd98765432  | 10      |        | 9     |
-----------------------------------------
| ef24681357  | 11      | 13     |       |
-----------------------------------------
```

Remarquez que les cases vides peuvent être représentées par l'absence d'une valeur (comme pour la première case de la deuxième ligne) ou tout simplement omises dans le fichier CSV si elle apparaît en fin de ligne (comme la dernière case).

2 Exercice 1 : Manipulation de fichier CSV

On représente le contenu d'un fichier CSV par la liste de ces lignes. Une ligne est elle-même la liste de ses valeurs (des chaînes de caractères).

```
type csv_content = string list list
```

1. Définissez une fonction `input_lines : in_channel -> string list` qui renvoie la liste des lignes du canal d'entrée passé en paramètre.
2. Définissez une fonction `read_csv : string -> csv_content` qui lit et renvoie le contenu d'un fichier CSV dont le nom est passé en paramètre.
3. Définissez une fonction `join : string -> string list -> string` telle que `join sep l` concatène les éléments de `l` séparés par le séparateur `sep`.

4. Définissez une fonction `write_csv : csv_content -> string -> unit` telle que `write_css c name` écrit `c` dans le fichier appelé `name` sous un format CSV.
5. Définissez une fonction `max_list : int list -> int list -> int list` qui renvoie une liste dont l'élément d'indice `i` est le maximum des éléments du même indice dans les listes passées en paramètres. Si une des listes est plus petite que l'autre, on considère que les éléments manquants sont égaux à 0.
6. Définissez une fonction `print_cell : int -> string -> unit` telle que `print_cell n str` affiche un espace, la chaîne `str` sur `n` caractères (on complète par des espaces si nécessaire), un espace et une barre. Par exemple, `print_cell 5 "foo"` affiche la chaîne `" foo |"`.
7. Définissez une fonction `print_line : int list -> string list -> unit` qui affiche le contenu d'une ligne. La première liste correspond aux tailles des colonnes et la deuxième au contenu des cellules. Par exemple, `print_line [5; 3] ["foo"; "bar"]` affiche la chaîne `"| foo | bar |"`.
8. Définissez une fonction `print_csv : csv_content -> unit` qui affiche le contenu d'un fichier CSV comme proposé dans l'exemple introductif.

3 Exercice 2 : Analyse d'un fichier CSV

On cherche à analyser des fichiers CSV dans un format particulier. La première ligne contient des intitulés. La deuxième ligne contient des coefficients (sauf pour la première cellule) correspondant à la pondération d'un examen pour le calcul de la moyenne d'une UE. Les lignes suivantes correspondent à des étudiants composés d'un identifiant et d'un certain nombre de notes. On cherche à obtenir le contenu d'un tel fichier comme un enregistrement `data` :

```
type student = {
  id : string;
  grades : float option list; (* None correspond à ABI *)
}

type data = {
  header : string list;
  coeff : float list;
  students : student list;
}
```

1. Définissez une exception `Invalid_format` qui sera levée dès que le fichier analysé ne correspond pas aux spécifications.
2. Définissez une fonction `parse_coeff : string list -> float list` qui prend en paramètre la ligne correspondant aux coefficients et les convertit en flottant.
3. Définissez une fonction `grades_from_strings : string list -> float option list` qui prend en paramètre la liste des notes et les convertit en flottants si possible. Toute note que l'on ne peut convertir en flottant devient la valeur `None`.
4. Définissez une fonction `normalize_grade : int -> float option list -> float option list` telle que `normalize_grade n l` renvoie la liste `l` complétée par des valeurs `None` pour qu'elle soit de taille `n`. Si la liste a une taille supérieure à `n`, on lève une exception `Invalid_format`.
5. Définissez une fonction `parse_student : int -> string list -> student` qui prend en paramètres un nombre de notes attendues et une ligne correspondant à un étudiant. Elle renvoie la représentation de l'étudiant associée.

6. Définissez une fonction `parse_csv : csv_content -> data` qui analyse le contenu d'un fichier CSV et en produit la représentation. Le nombre de notes attendues pour les étudiants est déduit du nombre de coefficients sur la deuxième ligne du fichier.

4 Exercice 3 : Modification d'un fichier CSV

On cherche maintenant à calculer la moyenne des étudiants présents dans un fichier CSV. Il faudra définir une fonction `process_file : string -> string -> unit` qui charge le contenu d'un fichier CSV, ajoute une colonne qui contient la moyenne de chaque étudiant, et l'écrit dans un nouveau fichier.

1. Définissez une fonction moyenne : `float list -> float option list -> float` qui prend en paramètres une liste des coefficients et une liste de notes pour renvoyer la moyenne pondérée de ces notes.
2. Définissez une fonction `process_student : float list -> student -> string list` qui prend en paramètres une liste de coefficients et un étudiant. Elle renvoie la ligne correspondant à l'étudiant. On y trouve son identifiant, ses notes (les valeurs `None` deviennent `"ABI"`) et sa moyenne.
3. Définissez la fonction `process_file` telle que `process_file name_in name_out` lit le fichier `name_in` et écrit dans le fichier `name_out`. Par exemple, le code suivant

```
let () = process_file "exemple.csv" "moyenne.csv"
let () = print_csv (read_csv "moyenne.csv")
```

affiche

```
-------------------------------------------------
| Identifiant | Partiel | Projet | Final | Moyenne |
-------------------------------------------------
|             | 0.3     | 0.3    | 0.4   |         |
-------------------------------------------------
| ab12345678  | 12.     | 15.    | 14.   | 13.7    |
-------------------------------------------------
| cd98765432  | 10.     | ABI    | 9.    | 6.6     |
-------------------------------------------------
| ef24681357  | 11.     | 13.    | ABI   | 7.2     |
-------------------------------------------------
```

Solution.

```
type csv_content = string list list

let rec input_lines ic =
  try
    let line = input_line ic in
    line :: input_lines ic
  with
  | End_of_file -> []

let read_csv name =
  let ic = open_in name in
  let content : csv_content = List.map (String.split_on_char ',') (input_lines ic) in
  close_in ic;
```

```
  content

let rec join sep = function
  | [] -> ""
  | [s] -> s
  | hd :: tl -> hd ^ sep ^ join sep tl

let write_csv (content : csv_content) name =
  let oc = open_out name in
  List.iter (fun line -> output_string oc (join "," line); output_string oc "\n") content;
  close_out oc

let rec max_list l1 l2 : int list = match l1, l2 with
  | l, [] | [], l -> l
  | hd1 :: tl1, hd2 :: tl2 -> max hd1 hd2 :: max_list tl1 tl2

let print_cell length str =
  print_string " ";
  print_string str;
  print_string (String.make (length - (String.length str)) ' ');
  print_string " |"

let print_line lengths line =
  let rec print_line_aux lengths line = match lengths, line with
    | [], _ -> print_newline ()
    | length :: lengths', [] ->
      print_cell length "";
      print_line_aux lengths' []
    | length :: lengths', str :: line' ->
      print_cell length str;
      print_line_aux lengths' line' in
  print_string "|";
  print_line_aux lengths line

let print_csv (content : csv_content) =
  let lengths_by_line = List.map (List.map String.length) content in
  let lengths = List.fold_left max_list [] lengths_by_line in
  let total_length = List.fold_left (fun sum length -> sum + length + 3) 1 lengths in
  let sep = (String.make total_length '-') ^ "\n" in
  print_string sep;
  List.iter (fun line -> print_line lengths line; print_string sep) content

type student = {
  id : string;
  grades : float option list;
}

type data = {
  header : string list;
  coeff : float list;
  students : student list;
}
```

```
exception Invalid_format

let parse_coeff = function
  | _ :: tl -> (try List.map float_of_string tl with _ -> raise Invalid_format)
  | _ -> raise Invalid_format

let rec grades_from_strings grades =
  List.map (fun str -> try Some (float_of_string str) with Failure _ -> None) grades

let rec normalize_grades n grades = match n, grades with
  | n, [] -> List.init n (fun _ -> None)
  | 0, _ -> raise Invalid_format
  | n, hd :: tl -> hd :: normalize_grades (n - 1) tl

let parse_student n = function
  | id :: grades -> {id; grades = normalize_grades n (grades_from_strings grades)}
  | _ -> raise Invalid_format

let parse_csv : csv_content -> data = function
  | header :: coeff_line :: student_lines ->
    let coeff  = parse_coeff coeff_line in
    let n = List.length coeff in
    let students = List.map (parse_student n) student_lines in
    {header; coeff; students}
  | _ -> raise Invalid_format

let rec moyenne coeff grades = match coeff, grades with
  | [], [] -> 0.0
  | _ :: coeff', None :: grades' -> moyenne coeff' grades'
  | c :: coeff', Some grade :: grades' -> c *. grade +. moyenne coeff' grades'
  | _ -> assert false

let process_student coeff {id; grades} =
  let grades_list = List.map (function
      | Some grade -> string_of_float grade
      | None -> "ABI") grades in
  id :: grades_list @ [string_of_float (moyenne coeff grades)]

let process_file name_in name_out =
  let content = read_csv name_in in
  let {header; coeff; students} = parse_csv content in
  let students_list = List.map (process_student coeff) students in
  let content' = (header @ ["Moyenne"]) ::
                 ("" :: (List.map string_of_float coeff)) :: students_list in
  write_csv content' name_out
```

□