- TP noté -

Le but de ce TP noté est de coder et décoder des messages en Morse.

Quelques remarques avant de commencer :

- Les types des fonctions demandées sont toujours donnés dans la question.
- Pensez à utiliser les fonctions des modules `List` et `String` (vous trouverez la documentation de ces modules dans le dossier `SUJET`).
- Vous avez toujours le droit de (et souvent intérêt à) ré-utiliser les fonctions écrites aux questions précédentes.
- Un fichier `morse_skeleton.ml` est fourni pour écrire vos fonctions. Il contient déjà les types donnés dans le sujet et quelques fonctions utiles. **Vous devrez obligatoirement le déposer dans le dossier `EXAM` déjà existant.**
- Le fichier de type `.ml` que vous rendez ne devra pas provoquer d'erreur pour `ocaml`.

1 Fonctions auxiliaires

Dans cette partie, on cherche à écrire des fonctions simples qui pourront être réutilisées dans la suite du TP et qui ne sont pas spécifiques au traitement des messages en Morse.

Question 1 Écrire la fonction `char_list_of_string` qui prend en paramètre une chaîne de caractères et renvoie la liste des caractères qui la composent (rappel : la fonction `String.get` permet d'accéder au différents caractères qui composent la chaîne).

```
(* val char_list_of_string : string -> char list = <fun> *)

char_list_of_string "pinguin";;
- : char list = ['p'; 'i'; 'n'; 'g'; 'u'; 'i'; 'n']
```

Question 2 Écrire la fonction `string_of_char_list` qui prend en paramètre une liste de caractères et renvoie la chaîne de caractères qu'ils forment. Dans les fichier `morse_skeleton.ml`, vous disposez d'une fonction `string_of_char` qui transforme un caractère en chaîne.

```
(* val string_of_char_list : char list -> string = <fun> *)

string_of_char_list ['p'; 'a'; 'n'; 'g'; 'o'; 'l'; 'i'; 'n'];;
- : string = "pangolin"
```

Question 3 Écrire la fonction `join_strings_with_string` qui prend en paramètre un délimiteur (de type `string`) et une liste de chaînes de caractères et renvoie la chaîne de caractères obtenue en joignant toutes les chaînes de la liste à l'aide du délimiteur.

```
(* val join_strings_with_string : string -> string list -> string = <fun> *)

join_strings_with_string " - " ["cat";"dog";"bird";"fly"];;
- : string = "cat - dog - bird - fly"
```

Question 4 Écrire la fonction `join_chars_with_string` qui fait la même chose que la fonction précédente mais avec une liste de caractères.

```
(* val join_chars_with_string : string -> char list -> string = <fun> *)

join_chars_with_string "*" ['p'; 'a'; 'n'; 'g'; 'o'; 'l'; 'i'; 'n'];;
- : string = "p*a*n*g*o*l*i*n"
```

Question 5 Écrire la fonction `contains_only` qui prend en paramètre deux listes `L1` et `L2` et vérifie (renvoie un booléen) que `L1` ne contient que des éléments qui sont présents dans `L2`.

```
(* val contains_only : 'a list -> 'a list -> bool = <fun> *)

contains_only ['b';'a';'c'] ['a';'b';'c';'d'];;
- : bool = true
contains_only ['b';'i';'c'] ['a';'b';'c';'d'];;
- : bool = false
```

Question 6 Écrire la fonction `fzip` qui prend en paramètre une fonction `f` à deux paramètres et deux listes `L1` et `L2` et renvoie la liste obtenue en appliquant `f` aux éléments de mêmes indices dans `L1` et `L2`. Si une des deux listes est plus longue que l'autre, les éléments surnuméraires ne sont pas traités.

Remarque : il existe une fonction similaire en caml (`map2`), mais vous n'avez pas le droit de l'utiliser pour cette question.

```
(* val fzip : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list = <fun> *)

fzip (+) [1;2] [5;6;7];;
- : int list = [6; 8]
fzip (+) [1;2;3] [5;6;7];;
- : int list = [6; 8; 10]
```

Question 7 Écrire la fonction `zip` qui prend en paramètre deux listes `L1` et `L2` et renvoie la liste obtenue en formant des couples avec les éléments de mêmes indices dans `L1` et `L2`. Si une des deux listes est plus longue que l'autre, les éléments surnuméraires ne sont pas traités.

Remarque : il existe une fonction similaire en caml (`combine`), mais vous n'avez pas le droit de l'utiliser pour cette question.

```
(* val zip : 'a list -> 'b list -> ('a * 'b) list = <fun> *)

zip [1;2] [5;6;7];;
- : (int * int) list = [(1, 5); (2, 6)]
```

Question 8 Écrire la fonction `unzip` qui prend en paramètre une liste de couples `L` et renvoie un couple de listes telles que la première liste contient les premiers éléments des couples de `L` et la seconde liste contient leurs seconds éléments.

Remarque : il existe une fonction similaire en caml (`split`), mais vous n'avez pas le droit de l'utiliser pour cette question.

```
(* val unzip : ('a * 'b) list -> 'a list * 'b list = <fun> *)

unzip (zip [1;2;3] [5;6;7]);;
- : int list * int list = ([1; 2; 3], [5; 6; 7])
```

2 Codage d'un message en Morse

Dans cette partie, on souhaite coder des messages (chaînes de caractères) en Morse. Le code Morse est défini à l'aide de trois symboles : un trait (*dash*), un point (*dot*) et un espace (*blank*). Pour coder un message en Morse, chaque lettre du message original est traduite en une lettre Morse qui est une liste de plusieurs symboles point ou trait. On peut ainsi représenter un texte en Morse comme une liste de lettres Morse, les mots étant séparés par une liste qui contient un espace. Voici les types correspondants. La liste d'association `morse_translation` (complète dans le fichier morse_skeleton.ml) fournit, quant à elle, le dictionnaire permettant de traduire les caractères alpha-numériques en lettres Morse.

```
type morse = Dash | Dot | Blank | NotAMorseSymbol

type morse_letter = morse list

(* val blank_translation : (char * morse list) list *)
let blank_translation : (char * morse_letter) list =
  [' ', [Blank]]

(* val num_translation : (char * morse list) list *)
let num_translation : (char * morse_letter) list =
  [ '1', [Dot; Dash; Dash; Dash; Dash];
    '2', [Dot; Dot; Dash; Dash; Dash];
    ...
    '0', [Dash; Dash; Dash; Dash; Dash]]

(* val char_translation : (char * morse list) list *)
let char_translation : (char * morse_letter) list =
  [ 'a', [Dot; Dash];
    'b', [Dash; Dot; Dot; Dot];
    ...
    'z', [Dash; Dash; Dot; Dot]]

(* morse code for chars 'a', 'b', .., 'z', '0', '1', ..., '9' *)
let morse_translation = blank_translation @ num_translation @ char_translation;;
```

Question 9 Écrire la fonction `check_msg` qui prend en paramètre une chaîne de caractères et vérifie qu'elle n'utilise que des caractères présents dans `morse_translation` (sans recréer à la main la liste des caractères).

```
(* val check_msg : string -> bool = <fun> *)

check_msg "happy otter day";;
- : bool = true
check_msg "Happy Otter Day!";;
- : bool = false
```

Question 10 Écrire la fonction `morse_letter_of_char` qui prend en paramètre un caractère et renvoie sa traduction en lettre Morse (en utilisant la liste d'association `morse_translation`).

```
(* val morse_letter_of_char : char -> morse_letter = <fun> *)

morse_letter_of_char 'a';;
- : morse_letter = [Dot; Dash]
```

Question 11 Écrire la fonction `char_of_morse_letter` qui prend en paramètre une lettre Morse et renvoie sa traduction en caractère (en utilisant `morse_translation`).

```
(* val char_of_morse_letter : morse_letter -> char = <fun> *)

char_of_morse_letter (morse_letter_of_char 'a');;
- : char = 'a'
```

Question 12 Écrire la fonction `morse_code` qui prend en paramètre un message qui n'utilise que des caractères présents dans `morse_translation` et renvoie son codage en morse.

```
(* val morse_code : string -> morse_letter list = <fun> *)

let msg = "happy otter day";;
let morse_code_example = morse_code msg;;
val morse_code_example : morse_letter list =
[[Dot; Dot; Dot; Dot]; [Dot; Dash]; [Dot; Dash; Dash; Dot]; [Dot; Dash; Dash; Dot];
 [Dash; Dot; Dash; Dash]; [Blank]; [Dash; Dash; Dash]; [Dash]; [Dash]; [Dot];
 [Dot; Dash; Dot]; [Blank]; [Dash; Dot; Dot]; [Dot; Dash]; [Dash; Dot; Dash; Dash]]
```

En réalité, un message en morse n'est pas une liste de listes (lettres Morse), mais une simple suite de sons : des traits (son long), des points (son court) et des silences. Les sons qui forment une lettre sont séparés par un seul silence (`SymbSpace`), les lettres sont séparées par trois silences (`LetterSpace`) et les mots par sept silences (`WordSpace`). On définit donc un nouveau type de Morse comme suit.

```
type morse_rendering = RDash | RDot | SymbSpace | LetterSpace | WordSpace| NotAMorseRenderingSymbol
```

Question 13 Écrire la fonction `render_letter_code` qui prend en paramètre une lettre Morse et renvoie sa traduction dans le type morse_rendering.

```
(* val render_letter_code : morse_letter -> morse_rendering list = <fun> *)

render_letter_code (morse_letter_of_char 'b');;
- : morse_rendering list =
[RDash; SymbSpace; RDot; SymbSpace; RDot; SymbSpace; RDot]
```

Question 14 Écrire la fonction `render_morse_code` qui prend en paramètre un message codé en Morse et renvoie sa traduction dans le type morse_rendering.

```
(* val render_morse_code : morse_letter list -> morse_rendering list = <fun> *)

let msg = "happy otter day";;
let render_morse_translation_msg = render_morse_code morse_code_example;;
val render_morse_translation_msg : morse_rendering list =
  [RDot; SymbSpace; RDot; SymbSpace; RDot; SymbSpace; RDot; LetterSpace;
   RDot; SymbSpace; RDash; LetterSpace; RDot; SymbSpace; RDash; SymbSpace;
   RDash; SymbSpace; RDot; LetterSpace; RDot; SymbSpace; RDash; SymbSpace;
   RDash; SymbSpace; RDot; LetterSpace; RDash; SymbSpace; RDot; SymbSpace;
   RDash; SymbSpace; RDash; WordSpace; RDash; SymbSpace; RDash; SymbSpace;
   RDash; LetterSpace; RDash; LetterSpace; RDash; LetterSpace; RDot;
   LetterSpace; RDot; SymbSpace; RDash; SymbSpace; RDot; WordSpace; RDash;
   SymbSpace; RDot; SymbSpace; RDot; LetterSpace; RDot; SymbSpace; RDash;
   LetterSpace; RDash; SymbSpace; RDot; SymbSpace; RDash; SymbSpace; RDash]
```

3 Décodage d'un message en Morse

Important : dans cette dernière partie, vous ne devez pas écrire de fonctions récursives et uniquement utiliser des fonctions d'ordre supérieur (comme, par exemple, `List.fold_left` ou `List.map`).
Cependant, si vous ne pouvez pas avancer dans les questions autrement, vous pouvez répondre avec des fonctions récursives, mais vous n'aurez pas tous les points pour les questions concernées.

Nous allons procéder au décodage des messages Morse. Pour cela, vous allez commencer par écrire deux fonctions auxiliaires qui vous permettront d'effectuer le décodage plus facilement.

Question 15 Écrire la fonction `split_list` qui prend en paramètre un élément `s` et une liste `L` d'éléments du même type que `s` et renvoie une liste formée des sous-listes de `L` délimitées par `s`. On supposera que ni le premier, ni le dernier élément de `L` n'est `s`.

```
(* val split_list : 'a -> 'a list -> 'a list list = <fun> *)

split_list 0 [1; 0; 2; 3; 0; 0; 4];;
- : int list list = [[1]; [2; 3]; [4]]
```

Question 16 Écrire la fonction `join_list_with` qui prend en paramètre un élément `s` et une liste `L` de listes d'éléments du même type que `s` et renvoie une liste formée des éléments des sous-listes de `L` entre lesquelles on a intercalé `s`.

```
(* val join_list_with : 'a -> 'a list list -> 'a list = <fun> *)

join_list_with 0 [[1];[2;3];[4]];;
- : int list = [1; 0; 2; 3; 0; 4]
```

Question 17 Écrire la fonction `rendering_to_morse` qui prend en paramètre un symbole de type morse_rendering et renvoie son équivalent (s'il existe) dans le type `morse`. S'il n'a pas d'équivalent, on renvoie `NotAMorseSymbol`.

```
(* val rendering_to_morse : morse_rendering -> morse = <fun> *)

rendering_to_morse RDot;;
- : morse = Dot
```

Question 18 Écrire la fonction `render_morse_decode` qui prend en paramètre un message écrit en `morse_rendering` et renvoie sa traduction en Morse.

```
(* val render_morse_decode : morse_rendering list -> morse_letter list = <fun> *)

let morse_translation_msg = render_morse_decode render_morse_translation_msg;;
val morse_translation_msg : morse_letter list =
[[Dot; Dot; Dot; Dot]; [Dot; Dash]; [Dot; Dash; Dash; Dot]; [Dot; Dash; Dash; Dot];
 [Dash; Dot; Dash; Dash]; [Blank]; [Dash; Dash; Dash]; [Dash]; [Dash]; [Dot];
 [Dot; Dash; Dot]; [Blank]; [Dash; Dot; Dot]; [Dot; Dash]; [Dash; Dot; Dash; Dash]]
```

Question 19 Écrire la fonction `morse_decode` qui prend en paramètre un message écrit en `morse` et renvoie sa traduction sous forme de chaîne de caractères.

```
(* val morse_decode : morse_letter list -> string = <fun> *)

morse_decode morse_translation_msg;;
- : string = "happy otter day"
```