Université de Bordeaux **Licence S&TS Mention Informatique, S3**
4TINA01U **2019/2020**

Programmation fonctionnelle
TP noté du Mercredi 11 Décembre
Durée : 1h20.

Le barème est donné à titre **indicatif**.
Le sujet comporte 4 pages plus une annexe.

Consignes

1. Depuis `Moodle`, téléchargez et décompressez l'archive `NOM-PRENOM.zip` contenant le squelette du tp.
2. Renommez le répertoire `NOM-PRENOM` en remplaçant `NOM` et `PRENOM` par vos nom et prénom. Exemple : `TURING-ALAN`.
3. Placez-vous dans le répertoire ainsi renommé.

Ce répertoire contient 5 fichiers. Les fichiers que vous aurez à éditer sont les trois fichiers suivants, `dict-list.ml`, `dict.ml` et `dict-fun.ml`.

Les fichiers `dict-list.ml` et `dict-fun.ml` correspondent à deux implémentations différentes d'un même type abstrait `dict`. Le fichier `dict.ml` contiendra des fonctions utilisant le type abstrait `dict` et valables quelle que soit l'implémentation choisie.

4. Indiquez à nouveau vos nom et prénom et votre numéro de groupe dans chacun de ces trois fichiers.

Les parties à compléter sont indiquées par des commentaires avec trois astérisques : `(*** ... ***)`.

Les fichiers `dict-expr.ml` et `dict-test.ml` sont des aides pour tester votre code. Le premier contient des expressions à tester. Le second permet de tester que les expressions retournent bien la valeur attendue. Après avoir chargé une implémentation (`dict-list.ml` ou `dict-fun.ml`) suivie du fichier `dict.ml`, vous pouvez la tester en évaluant entièrement le fichier `dict-test.ml`.

5. Pensez à sauver régulièrement votre travail.
6. À la fin du tp,
   (a) Déposez les trois fichiers `dict-list.ml`, `dict.ml` et `dict-fun.ml` à l'endroit prévu sur Moodle.
   (b) Créez une archive `<NOM>-<PRENOM>.zip` à partir de votre répertoire et envoyez-la à votre chargée de cours avec comme sujet `4TINA01U TP`
   - Groupe A2 : `frederique.carrere@u-bordeaux.fr`
   - Groupes A1 et A5 : `irene.durand@u-bordeaux.fr`

Un ***dictionnaire*** est un type abstrait représentant un ensemble de couples (clé, valeur) et associé aux opérations suivantes :

- fabriquer un dictionnaire vide (`dict_empty`),
- consulter la valeur associée à une clé (`dict_find key dict`),
- ajouter une entrée (`dict_add key v dict`), en associant une valeur à une nouvelle clé,
- supprimer une entrée (`dict_remove key dict`), c'est-à-dire de supprimer une clé et sa valeur associée.

Nous allons tester deux implémentations de ce type abstrait.

1 Implémentation avec listes

Cette première implémentation utilise une liste de couples (clé, valeur) pour représenter un dictionnaire. Le code de cette section doit être mis dans le fichier `dict-list.ml`.

Exercice 1 *Définition (2 pts)*

1. Définir un type `('a, 'b) dict` permettant de représenter un dictionnaire par une liste de couples (clé, valeur).
2. Initialiser la variable `dict_empty` avec le dictionnaire vide.

Exercice 2 *Adjonction (3 pts)*

3. Écrire la fonction `dict_add key v dict` qui retourne le dictionnaire `dict` dans lequel on a ajouté le couple `(key, v)` . Si la clé `key` n'était pas dans le dictionnaire, l'appel `add key val dict` ajoute cette clé. Sinon, elle remplace la valeur anciennement associée à `key` par `val` .

Exercice 3 *Consultation (2 pts)*

On se servira du type `'a option` prédéfini par `OCaml` , que l'on peut réécrire comme :

```
type 'a option = Some of 'a | None
```

Par exemple, une valeur du type `int option` est soit `None` , soit `Some x` où `x` est un entier. L'idée est que ce type permet de représenter des valeurs optionnelles. Par exemple, `Some 3` représente l'entier 3 et `None` ne représente aucune valeur. Cela sera utile, par exemple, pour la fonction permettant de consulter la valeur associée à une clé dans un dictionnaire : si la clé n'existe pas dans le dictionnaire, la fonction retournera `None` .
Exemples :

```
 # Some 0;;
- : int option = Some 0
# Some 4;;
- : int option = Some 4
# None;;
- : 'a option = None
```

4. Écrire une fonction `dict_find : 'a -> ('a,'b) dict -> 'b option` telle que `dict_find key dict` retourne la valeur associée à la clé `key` dans le dictionnaire `dict` si cette clé existe, et `None` sinon.

Exemples :

```
# let dict = dict_add 1 "un" (dict_add 2 "deux" (dict_add 3 "trois" dict_empty));;
val dict : (int * string) list = [(3, "trois"); (2, "deux"); (1, "un")]
# dict_find 4 dict;;
- : string option = None
# dict_find 2 dict;;
- : string option = Some "deux"
```

Exercice 4 *Suppression (2 pts)*

5. Écrire une fonction `dict_remove : 'a -> ('a, 'b) dict -> ('a, 'b) dict` telle que `dict_remove key dict` renvoie le dictionnaire `dict` dans lequel on a enlevé la l'entrée correspondant à la clé `key`, si cette clé est présente dans `dict` et renvoie `dict` s'il ne contient pas la clé `key`.

2 Utilisation du type dictionnaire

Les exercices de cette section concernent des fonctions qui utilisent les opérations du type abstrait et qui seront valables dans les deux implémentations. Le code de cette section doit être mis dans le fichier `dict.ml`.

Exercice 5 *(2 pts)*

6. Écrire une fonction `dict_adds couples dict` qui prend en paramètre une liste de couples (clé, valeur) et qui ajoute au dictionnaire `dict` chacune de ces liaisons (clé, valeur).

Ainsi

```
dict_adds [(1, "un"); (2, "deux");  (3, "trois")] dict_empty
```

doit donner la même chose que

```
dict_add 1 "un" (dict_add 2 "deux" (dict_add 3 "trois" dict_empty))
```

Exercice 6 *(1 pts)*

7. Écrire une fonction `make_dict couples` qui construit un nouveau dictionnaire à partir de la listes de couples (clé, valeur) `couples`.

Exercice 7 *(1 pts)*

8. Écrire une fonction `find_word word dict` qui étant donné un dictionnaire dont les clés et les valeurs sont des chaînes de caractères retourne la chaîne associée à `word` si elle existe la concaténation des trois chaînes `"?"`, `word` et `"?"` [1] sinon.

Exemples :

```
# let fr_en = make_dict
      [("bleu", "blue"); ("le", "the"); ("ciel", "sky");
       ("jaune", "yellow"); ("rouge", "red") ; ("vert", "green")];;
val fr_en : (string * string) list =
  [("bleu", "blue"); ("le", "the"); ("ciel", "sky"); ("jaune", "yellow");
```

1. soit `"?" ^ word ^ "?"`

```
   ("rouge", "red"); ("vert", "green")]
# let _ = find_word "ciel" fr_en;;
- : string = "sky"
# let _ = find_word "est" fr_en;;
- : string = "?est?"
```

Exercice 8 *(1 pts)*

9. Écrire une fonction `translate words dict` qui étant donné une phrase représentée par une liste de mots retourne la phrase traduite. Les mots inconnus dans le dictionnaire doivent être remplacés par avec « ?word ? ».

Exemples :

```
# translate ["le"; "ciel"; "est"; "bleu"] fr_en;;
- : string list = ["the"; "sky"; "?est?"; "blue"]
```

3 Implémentation avec des fonctions

On considère une deuxième implémentation dans laquelle un dictionnaire est représenté par une fonction de type `'a -> 'b option` où `'a` est le type des clés et `'b` le type des valeurs. Le code de cette section doit être mis dans le fichier `dict-fun.ml`.

Exercice 9 *(6 pts)*

10. Définir un type `('a, 'b) dict` permettant de représenter un dictionnaire par une fonction de type `'a -> 'b option`.

11. Initialiser la variable `dict_empty` avec le dictionnaire vide.

12. Implémenter l'opération `dict_add`.

13. Implémenter l'opération `dict_find`.

14. Implémenter l'opération `dict_remove`.

Annexe

```
let dict = dict_empty
let dict = dict_add 0 "zero" (dict_add 1 "un" dict)
let _ = dict_find 1 dict
let _ = dict_find 2 dict
let _ = dict_find 0 (dict_remove 0 dict)
let _ = dict_find 1 (dict_remove 0 dict)
let dict = dict_add 3 "trois" (dict_add 4 "quatre" dict)

let fr_en = make_dict
              [("bleu", "blue"); ("le", "the"); ("ciel", "sky");
               ("jaune", "yellow"); ("rouge", "red"); ("vert", "green")]

let _ = find_word "ciel" fr_en
let _ = find_word "est" fr_en
let words = ["le"; "ciel"; "est"; "bleu"]
let _ = translate words fr_en
```

FIG. 1: Fichier `dict-expr.ml`

```
let dict = make_dict [(1, "un"); (2, "deux"); (3, "trois"); (4, "quatre")]

let _ = assert(dict_find 1 dict = Some "un")
let _ = assert(dict_find 2 dict = Some "deux")
let _ = assert(dict_find 3 dict = Some "trois")
let _ = assert(dict_find 4 dict = Some "quatre")
let _ = assert(dict_find 5 dict = None)
let _ = assert(None = dict_find 4 (dict_remove 4 dict))
let _ = assert(Some "cinq" = dict_find 5 (dict_add 5 "cinq" dict))
```

FIG. 2: Fichier `dict-test.ml`