Ordre supérieur

1 Boite à outils

Rappels et fonctions utiles

Les fonctions que vous pouvez utiliser sont indiquées dans les exemples ci-dessous :

```
# (int_of_char 'A', int_of_char 'Z') ;;
 - : int * int = (65, 90)

# (char_of_int 97, char_of_int 122);;
 - : char * char = ('a', 'z')

# Char.escaped 'A' ;;
 string = "A"

#  'F' > 'A' && 'F' < 'Z';;
 - : bool = true
```

```
# let s = "a string" ;;
 val s : string = "a string"

# String.length s ;;
 - : int = 8

# (s.[0], s.[7]) ;;
 - : char * char = ('a', 'g')
```

1.1 `char` and `string`

char

1. Écrire la fonction `char_type c` qui retourne une chaîne correspondant au type du caratère c : `"lower"` ou `"upper"` si `c` est une lettre minuscule ou majuscule, `"other"` sinon.

```
 val char_type : char -> string = <fun>
# (char_type 'a', char_type 'G', char_type ' ') ;;
 - : string * string * string = ("lower", "upper", "other")
```

2. Écrire la fonction `uppercase` qui transforme une lettre minuscule en majuscule, retourne le caractère inchangé sinon.

```
 val uppercase : char -> char = <fun>
# (uppercase 'a', uppercase 'G', uppercase ' ') ;;
 - : char * char * char = ('A', 'G', ' ')
```

3. Écrire la fonction `lowercase` qui transforme une lettre majuscule en minuscule, retourne le caractère inchangé sinon.

```
 val lowercase : char -> char = <fun>
# (lowercase 'A', lowercase 'h', lowercase ',') ;;
 - : char * char * char = ('a', 'h', ',')
```

4. Écrire la fonction `swap_alpha c` qui retourne le symétrique dans l'alphabet de `c` si `c` est une lettre.

```
 val swap_alpha : char -> char = <fun>
# (swap_alpha 'F', swap_alpha 'c', swap_alpha ' ') ;;
 - : char * char * char = ('U', 'x', ' ')
```

5. Écrire la fonction `rotn n c` qui applique un décalage de `n` (à droite si `n` positif) au caractère `c` si `c` est une lettre.

```
 val rotn: int -> char -> char = <fun>
# (rotn 3 's', rotn 13 'A', rotn (-5) 'z', rotn 5 'u', rotn 5 ' ');;
 - : char * char * char * char * char = ('v', 'N', 'u', 'z', ' ')
```

Rappel : application partielle

```
# let rot13 = rotn 13 ;;
 val rot13 : char -> char = <fun>

# rot13 'A' ;;
 - : char = 'N'
```

`string` ↔ `list`

1. Écrire la fonction `string_of_list` qui transforme une liste de caractères en une chaîne.

```
val string_of_list : char list -> string = <fun>

# string_of_list ['I'; 'm'; 'p'; 'l'; 'o'; 'd'; 'e'] ;;
- : string = "Implode"
```

2. Écrire la fonction `list_of_string` qui transforme une chaîne en une liste de caractères.

```
val list_of_string : string -> char list = <fun>

# list_of_string "Explode" ;;
- : char list = ['E'; 'x'; 'p'; 'l'; 'o'; 'd'; 'e']
```

1.2 Ordre supérieur

1. Écrire la fonction `map f` $[a_1;\ \ldots;\ a_n]$ qui applique la fonction `f` à $a_1;\ldots;\ a_n$ et construit la liste `[f` a_1`; f` a_2`;...; f` a_n`]` avex les valeurs retournées par `f`.

```
val map : ('a -> 'b) -> 'a list -> 'b list
```

Utiliser `map` pour écrire la fonction `uppercase_list` qui convertit toutes les lettres minuscules d'une liste en majuscules.
Écrire également la fonction `lowercase_list`.

```
val uppercase_list : char list -> char list = <fun>
val lowercase_list : char list -> char list = <fun>

# uppercase_list ['a'; 'B'; 'c'; 'd'; 'e'; ' '; '0'] ;;
- : char list = ['A'; 'B'; 'C'; 'D'; 'E'; ' '; '0']
# lowercase_list ['A'; 'b'; 'C'; 'D'; 'E'; ' '; ','] ;;
- : char list = ['a'; 'b'; 'c'; 'd'; 'e'; ' '; ',']
```

2. Écrire la fonction `iter f` $[a_1;\ \ldots;\ a_n]$ qui applique la fonction `f` à $a_1;\ldots;\ a_n$séquentiellement. iter f $[a_1 ; \ldots ; a_n]$ est équivalent à `begin f` a_1`; f` a_2`;...; f` a_n`; () end`.

```
val iter : ('a -> unit) -> 'a list -> unit
```

3. Écrire la fonction `map2 f` $[a_1; a_2; \cdots; a_n]$ $[b_1; b_2; \cdots; b_n]$ qui construit la liste : $[f\ a_1\ b_1\ ;\ f\ a_2\ b_2\ ;\ \cdots\ ;\ f\ a_n\ b_n]$. Elle déclenche une exception si les deux listes sont de longueurs différentes.

2 Un peu d'ordre supérieur : Chiffrements

2.1 Chiffre de César

En cryptographie, le chiffrement par décalage, aussi connu comme le chiffre de César ou le code de César, est une méthode de chiffrement très simple utilisée par Jules César dans ses correspondances secrètes.Le texte chiffré s'obtient en remplaçant chaque lettre du texte clair original par une lettre à distance fixe, toujours du même côté, dans l'ordre de l'alphabet. Pour les dernières lettres (dans le cas d'un décalage à droite, utilisé ici), on reprend au début. Par exemple avec un décalage de `3` vers la droite, `A` est remplacé par `D`, `B` devient `E`, et ainsi jusqu'à `W` qui devient `Z`, puis `X` devient `A` etc. Il s'agit d'une permutation circulaire de l'alphabet.

wikipédia

1. En utilisant les fonctions précédentes, écrire la fonction `caesar_encode n s` qui applique le chiffrement de César avec un decalage `n` à la chaîne `s`.

```
val caesar_encode : int -> string -> string = <fun>

# caesar_encode 3 "Krisboul" ;;
- : string = "Nulverxo"

# caesar_encode 13 "Chiffrement" ;;
- : string = "Puvsserzrag"
```

2. En utilisant les fonctions précédentes, écrire la fonction `caesar_decode n s` qui décode la chaîne `s` chiffrée avec un decalage `n`.

```
val caesar_decode : int -> string -> string = <fun>

# caesar_decode 3 "Nulverxo";;
 - : string = "Krisboul"
# caesar_decode 5 "Ymjxj antqjsy ijfiqnsjx mfaj antqjsy jsix." ;;
 - : string = ...
```

2.2 Chiffre de Vigenère

Le chiffre de Vigenère est un système de chiffrement polyalphabétique, c'est un chiffrement par substitution, mais une même lettre du message clair peut, suivant sa position dans celui-ci, être remplacée par des lettres différentes, contrairement à un système de chiffrement monoalphabétique comme le chiffre de César. Ce chiffrement introduit la notion de clé. Une clé se présente généralement sous la forme d'un mot ou d'une phrase. Pour pouvoir chiffrer notre texte, à chaque caractère nous utilisons une lettre de la clé pour effectuer la substitution. Évidemment, plus la clé sera longue et variée et mieux le texte sera chiffré.

wikipédia [1]

Un exemple avec "Vigenere cipher" la chaîne à coder et "abc" la clé :

```
string to encode :  Vigenere Cipher
key, repeated :     abcabcab cabcab
                    ---------------
result :            Vjieogrf Eiqjes
```

1. Écrire les fonctions `char_encode_vigen c ckey` et `char_decode_vigen c ckey` qui encode (respectivement décode) une lettre `c` du message en clair (respectivement chiffré) avec une lettre de la clé `ckey`.

```
val char_encode_vigen : char -> char -> char = <fun>
val char_decode_vigene : char -> char -> char = <fun>
# char_encode_vigen 'T' 'E' ;;
 - : char = 'X'
# char_encode_vigen 'T' 'M' ;;
 - : char = 'F'
# char_decode_vigen 'V' 'L' ;;
 - : char = 'K'
```

2. Écrire la fonction `gen_key_list list key` qui génère la liste des caractères de la clé `key` allignés sur les lettres de `list`.

```
val gen_key_list : char list -> char list -> char list = <fun>

# gen_key_list
['V'; 'i'; 'g'; 'e'; 'n'; 'e'; 'r'; 'e'; ' '; 'C'; 'i'; 'p'; 'h'; 'e'; 'r'] ['K'; 'e'; 'y'];;
- : char list =
['K'; 'e'; 'y'; 'K'; 'e'; 'y'; 'K'; 'e'; ' '; 'y'; 'K'; 'e'; 'y'; 'K'; 'e']
```

3. Écrire la fonction `vigenere` qui prend en paramètre la fonction d'encodage ou de décodage, la clé et le message à chiffrer ou à déchiffrer.

```
val vigenere : (char -> char -> char) -> string -> string -> string = <fun>

# vigenere char_encode_vigen "abc" "Vigenere Cipher" ;;
 - : string = "Vjieogrf Eiqjes"

# vigenere char_decode_vigen "Caml" "Vo Dpeuddg ie okvuyg" ;;
 - : string = "..."
# vigenere char_decode_vigen "ACDC" "Wjb wgnb fecokngv?" ;;
 - : string = "..."
```

1. https://fr.wikipedia.org/wiki/Chiffre_de_Vigenère

3 Build House : l'ordre supérieur au travail

3.1 Travail à la chaîne

Les listes de caractères suivantes représentent les différentes matières premières nécessaires à la construction de notre maison.

```
let sand = ['B' ; 'b' ; 'L' ; 'r' ; 'B'; 'B' ; '\n'] ;;
let water = ['y' ; 'o' ; 'y' ; 'y' ; 'i'; 'y'; '\n'] ;;
let brick = ['y' ; 's' ; 's' ; 's' ; 's'; 'e'; '\n'] ;;
let wood = ['h' ; 'r' ; 'w' ; 'w' ; 'r'; 'h'; '\n'] ;;
let coca = ['s' ; 'l' ; 'r' ; 'x' ; 'l'; 's'; '\n'] ;;
```

Notre maison étant l'assemblage de ces matières premières, nous allons en faire une liste.

```
# let house = [sand ; water ; brick ; wood ; coca];;
 val house : char list list = ...
```

Maintenant, il faut indiquer à nos ouvriers dans quel ordre il vont travailler afin que notre maison soit bien construite.

```
# let workers = [lowercase ; swap_alpha ; rot13 ; rotn 5 ; rotn 20];;
 val workers : (char -> char) list = [<fun>; <fun>; <fun>; <fun>; <fun>]
```

3.1.1 Au travail !

Maintenant que nos ouvriers savent dans quel ordre il doivent travailler, il ne vous reste plus qu'à leur dire comment travailler. Vous allez créer une fonction `chain` qui prend en paramètre une liste de fonctions (genre `workers`) et une liste de listes de caractères (genre `house`) et qui retourne la liste de listes de caractères, auxquels on aura appliqué une fonction de la liste de fonctions à chaque liste. Pour rendre tout cela plus clair regardez l'exemple suivant :

```
chain [f1, f2] [['a' ; 'c'; 'd' ; 'c'] ; ['s'; 'u'; 'p']]
-> [[f1 'a'; f1 'c'; f1 'd'; f1 'c'] ; [f2 's'; f2 'u'; f2 'p']]
```

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

Vous l'aurez compris, il faut appliquer cette fonction à `workers` et `house`, afin de récupérer la nouvelle liste de listes qui représentera les fondations de notre maison.

```
# let fundations = chain workers house;;
 val fundations : char list list = ...
```

3.1.2 Finalisation

Vos fondations étant complètement finies, il ne vous reste plus qu'à bâtir (afficher) votre maison. Par exemple, voici les caractères à afficher en fonction du type element rencontré :

```
'm' (wall)       ->  '|'
'f' (floor)  ->  '_'
'l' (left roof) ->  '/'
'r' (right roof) ->  '\'
'w' (window) ->  '+'
'b' (blanck) ->  ' '
'\n' (end of ground) ->  '\n'
```

Écrire la fonction `print_house f house` avec `f` la fonction d'affichage des matériaux.

```
val val print_house : ('a -> 'b) -> 'a list list -> unit = <fun>
```

Continuez à utiliser l'ordre supérieur : iter est votre ami :)