

TP n°3

Objectifs du TP :

- Fonctions d'ordre supérieur
- Fonctions impures

Exercice n°1 : matrices fonctionnelles

Soit une matrice M de dimension $L \times C$ représentée par une fonction binaire $f(i,j)$ qui retourne un couple $(\text{Bool}, \text{Int})$ de la façon suivante :

$$\begin{cases} (\text{True}, \text{Int}) & \text{si } 1 \leq i \leq L \text{ et } 1 \leq j \leq C \\ (\text{False}, 0) & \text{sinon} \end{cases}$$

Exemple : la matrice *6 par 5* ci-dessous

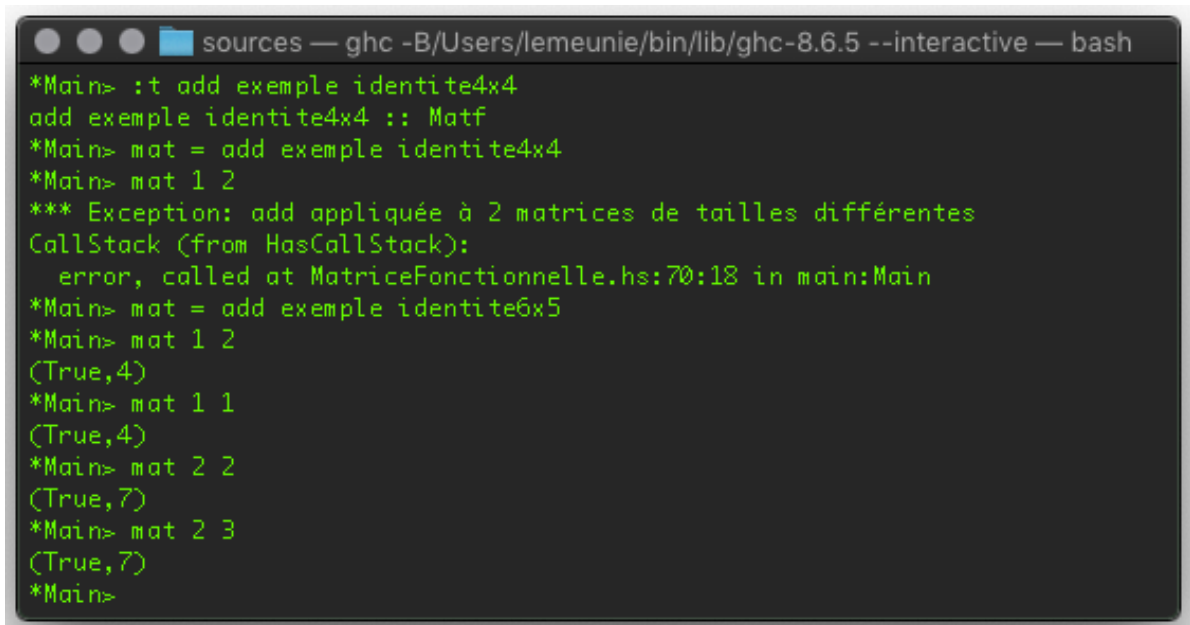
$$\begin{pmatrix} 3 & \dots & 7 \\ \vdots & \ddots & \vdots \\ 13 & \dots & 17 \end{pmatrix}$$

est représentée par la fonction *exemple* qui retourne :

$$\begin{cases} (\text{True}, 2 * i + j) & \text{si } 1 \leq i \leq 6 \text{ et } 1 \leq j \leq 5 \\ (\text{False}, 0) & \text{sinon} \end{cases}$$

- Ecrivez un type matrice fonctionnelle nommé *Matf*.
- Ecrivez la fonction *exemple*.
- Ecrivez la fonction *identite4x4* représentant une matrice 4×4 avec des 1 sur sa diagonale et des 0 partout ailleurs.
- Ecrivez la fonction *dims :: Matf -> (Int, Int)* qui retourne les dimensions de la matrice donnée en paramètre. Vous écrirez pour cela une fonction *nbLines :: Matf -> Int* qui retourne le nombre de ligne d'une matrice et *nbCols :: Matf -> Int* qui retourne le nombre de colonne d'une matrice.
- Ecrivez une fonction *cmpDims :: Matf -> Matf -> Bool* qui teste si deux matrices sont de même dimensions.

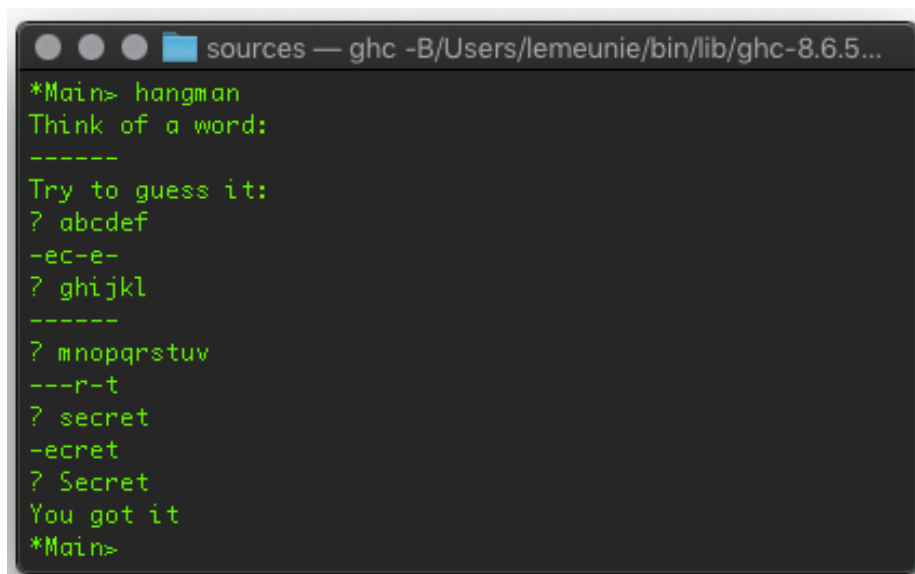
- f) Ecrivez une fonction `add :: Matf -> Matf -> Matf` qui « fait la somme » des deux matrices données en paramètre. Vérifiez que les deux matrices sont de même dimension sinon levez une exception.
- g) Ecrivez une matrice `identite6x5` et expliquez les résultats suivants :



```
sources — ghc -B/Users/lemeunie/bin/lib/ghc-8.6.5 --interactive — bash
*Main> :t add exemple identite4x4
add exemple identite4x4 :: Matf
*Main> mat = add exemple identite4x4
*Main> mat 1 2
*** Exception: add appliquée à 2 matrices de tailles différentes
CallStack (from HasCallStack):
  error, called at MatriceFonctionnelle.hs:70:18 in main:Main
*Main> mat = add exemple identite6x5
*Main> mat 1 2
(True,4)
*Main> mat 1 1
(True,4)
*Main> mat 2 2
(True,7)
*Main> mat 2 3
(True,7)
*Main>
```

Exercice n°2 : le jeu du pendu

Le jeu du pendu consiste à trouver un mot imaginé par l'adversaire. Dans votre version informatisée, un premier joueur imagine un mot puis le saisie au clavier. L'affichage devra alors être masqué. L'adverse va ensuite faire des propositions. Si la proposition échoue, il recommence mais le programme doit découvrir au fur et mesure les lettres correctes. Voici un exemple du jeu en action :



```
sources — ghc -B/Users/lemeunie/bin/lib/ghc-8.6.5...
*Main> hangman
Think of a word:
-----
Try to guess it:
? abcdef
-ec-e-
? ghijkl
-----
? mnopqrstuv
---r-t
? secret
-ecret
? Secret
You got it
*Main>
```

- a) Ecrivez une fonction *getCh :: IO Char* qui retournera le caractère saisi. L'écho doit être arrêté pendant la saisie. Pour cela utilisez la fonction *hSetEcho* du module *System.IO*.
- b) Ecrivez une fonction *sgetline :: IO String* qui retournera une chaîne saisie. A l'écran, chaque caractère saisi sera remplacé par le caractère '-'.
- c) Ecrivez une fonction *match :: String -> String -> String* qui découvre les caractères correctes de la chaîne à deviner.
Par exemples : *match "toto" "tata" == "t-t-"*
match "toto" "gogo" == "-o-o"
match "Nottingham" "Glasgow" == "-o----g-a-"
- d) Ecrivez la fonction récursive *play :: String -> IO ()* qui prend comme paramètre la chaîne à trouver et qui gère les propositions de l'adversaire jusqu'à ce qu'il est trouvé.

```
sources — ghc -B/Users/le...
*Main> play "toto"
? titi
t-t-
? tata
t-t-
? toto
You got it
*Main>
```

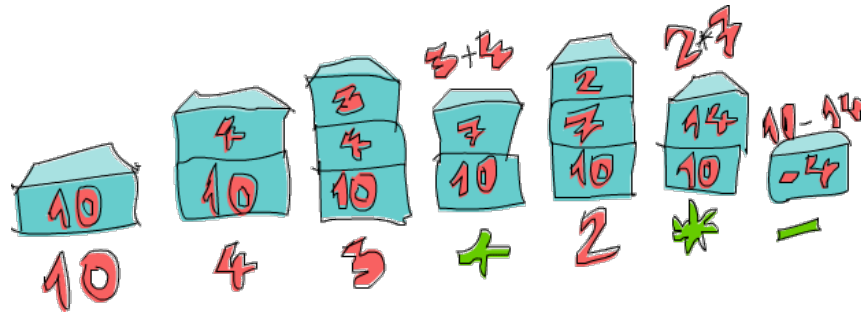
- e) Ecrivez la fonction *hangman :: IO ()* qui demande le mot secret puis fait jouer l'adversaire en appliquant la fonction *play*.
- f) Ecrivez la fonction principale *main* puis compilez votre programme avec le compilateur *ghc* et exécutez-le directement dans le terminal.

Exercice n°3 : expression NPI

La notation polonaise inverse (ou NPI) est une façon d'écrire des expressions algébriques en post-fixe ce qui évite de devoir utiliser des parenthèses.

Expression infixe habituelle	Expression NPI équivalente
10 - (4 + 3) * 2	10 4 3 + 2 * -

Une expression NPI est évaluée en utilisant une pile :



Vous allez écrire une fonction *evalNPI* d'évaluation d'une expression NPI. Cette fonction aura pour type *evalNPI :: (Num a, Read a) => String -> a*. On supposera que l'expression NPI est bien formée et sémantiquement exacte.

L'évaluation d'une expression NPI se fait en parcourant la liste des éléments de l'expression de la gauche vers la droite et pour chaque élément de l'empiler si l'élément est un nombre ou d'empiler le résultat du calcul si l'élément est un opérateur. Il s'agit donc d'un pliage d'une liste !

On utilisera une liste comme pile avec la tête de liste comme sommet.

- Etudiez la fonction *words :: String -> [String]*.
Que donne l'expression *words "10 4 3 + 2 * -"* ?
- Ecrivez une fonction d'évaluation d'un élément dans le contexte d'une pile.
Le type sera *evalElem :: (Num a, Read a) => [a] -> String -> [a]*.

Exemples : *evalElem [3, 4, 10] "+" == [7, 10]*
evalElem [7, 10] "2" == [2, 7, 10]

- Ecrivez la fonction de plis *evalNPI*. Utilisez la fonction *evalElem*.

Exemples : *evalNPI "10 4 3 + 2 * -" == -4*
evalNPI "2 3 +" == 5
*evalNPI "90 34 12 33 55 66 + * - +" == -3947*

- Ecrivez la fonction *evalNPI* sans parenthèse sous forme curriifiée.

Références web

Site officiel Haskell
<https://www.haskell.org>

Site des modules officiels Haskell
<https://downloads.haskell.org/~ghc/latest/docs/html/libraries/>