

**Examen du 19 décembre 2019**  
*Document de cours, TD et TP autorisés***Exercice n°1 (2 pts)**

Donnez l'implémentation de la fonction  $zip3 :: [a] \rightarrow [b] \rightarrow [c] \rightarrow [(a,b,c)]$

Exemples :  $zip3 [1,2,3] [4,5,6] [7,8,9] == [(1,4,7),(2,5,8),(3,6,9)]$   
 $zip3 [1,2] [4,5,6] [7,8] == [(1,4,7),(2,5,8)]$

**Exercice n°2 (3 pts)**

Donnez l'implémentation de la fonction  $unzip :: [(a,b)] \rightarrow ([a], [b])$ . Utilisez la fonction *foldr*.

Exemple :  $unzip [(1,4),(2,5),(3,6)] == ([1,2,3],[4,5,6])$

**Problème (15 pts)**

Il s'agit d'implémenter un jeu de dame (simplifié par rapport aux règles internationales). On n'implémentera pas tout le code nécessaire pour faire un logiciel complet, mais uniquement quelques fonctions de base.

Les questions peuvent être traitées indépendamment les unes des autres.

Les types utilisés seront les suivants :

```
data Joueur = JBlanc | JNoir

data Pion = Vide | Noir | Blanc | DameN | DameB
  deriving (Eq, Ord)

type Ligne = [Pion]
type Damier = [Ligne]

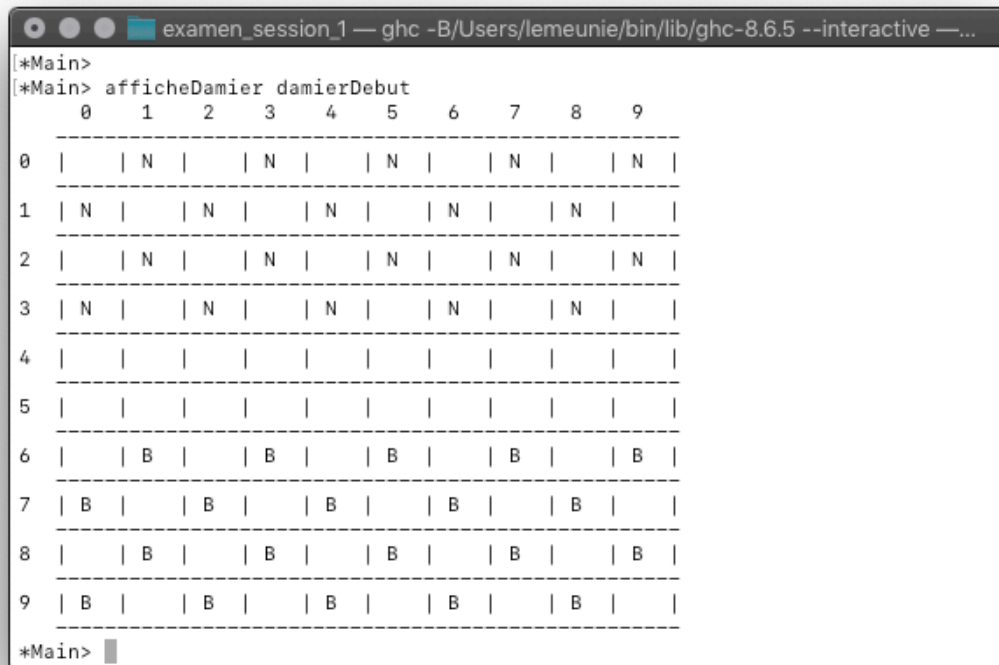
-- Case du damier : (ligne, colonne) avec début en (0,0) en haut à gauche
type Position = (Int,Int)
```

La taille du damier ne pouvant pas changer, on codera « en dur » les différentes dimensions.

La base blanche se trouve ligne 9 et la base noire se trouve ligne 0 (cf. image ci-après).

### Question n°1 (1 pt)

Ecrivez l'instanciation de la classe *Show* par le type *Pion* (cf. image ci-après).  
Les dames seront représentées par " DN " et " DB ".



```
examen_session_1 — ghc -B/Users/lemeunie/bin/lib/ghc-8.6.5 --interactive —...
[*Main>
[*Main> afficheDamier damierDebut
  0   1   2   3   4   5   6   7   8   9
  0 |   | N |   | N |   | N |   | N |   | N |
  1 | N |   | N |   | N |   | N |   | N |   |
  2 |   | N |   | N |   | N |   | N |   | N |
  3 | N |   | N |   | N |   | N |   | N |   |
  4 |   |   |   |   |   |   |   |   |   |   |
  5 |   |   |   |   |   |   |   |   |   |   |
  6 |   | B |   | B |   | B |   | B |   | B |
  7 | B |   | B |   | B |   | B |   | B |   |
  8 |   | B |   | B |   | B |   | B |   | B |
  9 | B |   | B |   | B |   | B |   | B |   |
[*Main> ]
```

### Question n°2 (1 pt)

Ecrivez la fonction *damierDebut* :: *Damier* qui retourne le damier de départ (cf. image ci-dessus).

### Question n°3 (1 pt)

Ecrivez la fonction *afficheLigne* :: *Ligne* -> *IO ()* qui affiche une ligne d'un damier (sans le numéro de ligne en début et sans les lignes horizontales de séparation).

### Question n°4 (2 pts)

Ecrivez la fonction *afficheDamier* :: *Damier* -> *IO ()* qui affiche le damier complet avec les numéros de colonne et de ligne comme dans l'image ci-dessus.

### Question n°5 (1 pt)

Ecrivez la fonction *donnePion* :: *Damier* -> *Position* -> *Pion* qui retourne le pion se trouvant à la position donnée. On suppose que la position donnée est valide ; il ne faut pas la tester.

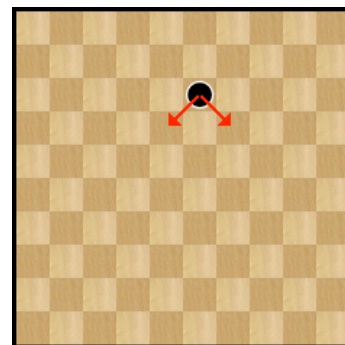
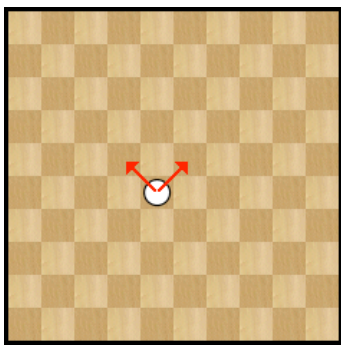
### Question n°6 (1 pt)

Ecrivez la fonction `posValide :: Position -> Bool` qui teste si la position donnée est valide (c'est-à-dire qu'elle ne sort pas des limites du damier)

### Question n°7 (2 pts)

Ecrivez la fonction `deplacementPionValide :: Damier -> Position -> Position -> [Position]`. Cette fonction teste si on peut déplacer un pion (blanc ou noir) d'une position donnée vers une nouvelle position donnée. Par convention, la fonction retourne une liste vide si le déplacement n'est pas possible. Sinon elle retourne la liste des deux positions successives du pion.

On respectera la règle suivante : un pion se déplace obligatoirement « vers l'avant », en diagonale, d'une case sur une case libre de la rangée suivante. Les pions blancs se déplacent vers la base noire. Les pions noirs se déplacent vers la base blanche.



Exemples : `deplacementPionValide damierDebut (6,1) (5,0) == [(6,1),(5,0)]`  
`deplacementPionValide damierDebut (6,1) (5,1) == []`

### Question n°8 (2 pts)

Vous disposez de la fonction suivante :

```
damePeutPrendre :: Damier -> Pion -> Pion -> Position -> Bool
damePeutPrendre d DameB Vide _ = True
damePeutPrendre d DameB Noir p = donnePion d p == Vide
damePeutPrendre d DameB DameN p = donnePion d p == Vide
damePeutPrendre d DameN Vide p = True
damePeutPrendre d DameN Blanc p = donnePion d p == Vide
damePeutPrendre d DameN DameB p = donnePion d p == Vide
damePeutPrendre _ _ _ _ = False
```

Ecrivez la fonction `posDiagonale :: Damier -> Pion -> Position -> Position -> [Position]` qui retourne la liste des positions en diagonale depuis une position de départ (2<sup>ème</sup> paramètre) vers une position d'arrivée (3<sup>ème</sup> paramètre). Si la position finale n'est pas atteignable depuis la position initiale, la liste retournée est vide. On suppose que la pièce de départ (1<sup>er</sup> paramètre) est une dame.  
**Attention** : la position de départ n'est pas incluse dans la liste résultat.

```

examen_session_1 — ghc -B/Users/lemeunie/bin/lib/...
*Main> afficheDamier d2
      0   1   2   3   4   5   6   7   8   9
0  |   |   |   |   |   |   |   |   |   |
1  |   |   |   |   |   |   |   |   |   |
2  |   |   |   |   |   |   |   |   |   |
3  |   |   |   |   |   |   |   |   |   |
4  |   |   |   |   |   |   N   |   |   |   |
5  |   |   |   |   |   |   |   |   |   |   |
6  |   |   |   |   |   |   |   |   |   |   |
7  |   |   |   |   |   |   |   |   |   |   |
8  |   |   |   |   |   |   |   |   |   |   |
9  | DB |   |   |   |   |   |   |   |   |   |

*Main> posDiagonale d2 DameB (9,0) (1,8)
[(8,1),(7,2),(6,3),(5,4),(4,5),(3,6),(2,7),(1,8)]
*Main>

```

### Question n°9 (2 pts)

Ecrivez la fonction *placePion* :: *Damier* -> *Pion* -> *Position* -> *Damier* qui place un pion (2<sup>ème</sup> paramètre) à la position (3<sup>ème</sup> paramètre) du damier (1<sup>er</sup> paramètre).

### Question n°10 (2 pts)

Ecrivez la fonction *priseEnDiagonale* :: *Damier* -> [*Position*] -> *Damier* qui exécute une prise par un pion ou par une dame ou un déplacement d'un pion ou d'une dame sur le damier.

**Attention : cette fonction n'est prévue que pour une liste de positions d'une même diagonale.**

La liste des positions (2<sup>ème</sup> paramètre) correspond aux positions de la diagonale de la pièce. Cette liste commence par la position initiale de la pièce et finit par la position finale de la pièce. Plusieurs positions intermédiaires peuvent être données si elles existent mais toutes les positions de la liste doivent être sur une même diagonale.

## Exemple1 :

```
examen_session_1 — ghc -B/Users/lemeunie/bin/lib/gh...
*Main>
*Main>
*Main> afficheDamier d3
      0 1 2 3 4 5 6 7 8 9
0 | | | | | | | | | |
1 | | | | | | | | | |
2 | | | | | | | | | |
3 | | | | | | | | | |
4 | | | | | N | | | |
5 | | | | B | | | | |
6 | | | B | | | | | |
7 | | | | | | | | | |
8 | | | | | | | | | |
9 | | | | | | | | | |
```

```
examen_session_1 — ghc -B/Users/lemeunie/bin/lib/gh...
*Main> d4 = priseEnDiagonale d3 [(5,4),(4,5),(3,6)]
*Main> afficheDamier d4
      0 1 2 3 4 5 6 7 8 9
0 | | | | | | | | | |
1 | | | | | | | | | |
2 | | | | | | | | | |
3 | | | | | B | | | |
4 | | | | | | | | | |
5 | | | | | | | | | |
6 | | | B | | | | | |
7 | | | | | | | | | |
8 | | | | | | | | | |
9 | | | | | | | | | |
*Main>
```

## Exemple2 :

```
examen_session_1 — ghc -B/Users/lemeunie/bin/lib/gh...
*Main> d1 = placePion damierVide DameB (9,0)
*Main> d2 = placePion d1 Noir (4,5)
*Main> afficheDamier d2
      0 1 2 3 4 5 6 7 8 9
0 | | | | | | | | | |
1 | | | | | | | | | |
2 | | | | | | | | | |
3 | | | | | | | | | |
4 | | | | N | | | | |
5 | | | | | | | | | |
6 | | | | | | | | | |
7 | | | | | | | | | |
8 | | | | | | | | | |
9 | DB | | | | | | | |
*Main>
```

```
examen_session_1 — ghc -B/Users/lemeunie/bin/lib/gh...
*Main> d3 = priseEnDiagonale d2 [(9,0),(8,1),(7,2),(6,3),
[(5,4),(4,5),(3,6),(2,7),(1,8)]
*Main> afficheDamier d3
      0 1 2 3 4 5 6 7 8 9
0 | | | | | | | | | |
1 | | | | | | DB | | |
2 | | | | | | | | | |
3 | | | | | | | | | |
4 | | | | | | | | | |
5 | | | | | | | | | |
6 | | | | | | | | | |
7 | | | | | | | | | |
8 | | | | | | | | | |
9 | | | | | | | | | |
*Main>
```