Programmation Fonctionnelle : TD2

Université de Tours

`Département informatique de Blois`

Expressions booléennes, prédicats, conditions, structures

*
* *

Appropriation du cours

Revenons sur le cours 2, qui se trouve sous Celene : testez en Ocaml les exemples ou exercices suivants :

- La fonction `divisiblePar` (définition/code et application à des arguments).
- La fonction `bissextile` (définition/code et application à des arguments).
- La fonction `stringTestDiv` (définition/code et application à des arguments).
- La fonction `secondeSuivante` (définition/code et application à des arguments), ***dans les 2 versions***, et en testant aussi une version dans laquelle vous mettez les motifs (pattern) dans l'ordre inverse pour le switch, afin de constater que c'est bien différent.
- Une fonction `add_Fraction` qui fait la somme de 2 fractions (définition et application à des arguments) avec définition d'un nouveau type `fraction` ***et*** sans définition de nouveau type, en utilisant un couple d'entiers $(n_1, n_2) \in \mathbb{Z} \times \mathbb{N}^*$.

Problème 1

On s'intéresse ici à la modélisation de l'égalité de deux fractions :

1. Écrire la spécification et le code d'une fonction `testEgalite1` de deux fractions à l'aide du type `fraction` défini précédemment.

   On rappelle que:

   $\frac{a}{b} = \frac{c}{d} \Leftrightarrow \frac{a}{b} - \frac{c}{d} = 0$
   $\Leftrightarrow \frac{ad-cb}{bd} = 0$
   $\Leftrightarrow ad - cb = 0$

   Signature:

   $$\begin{cases} \texttt{fraction} \times \texttt{fraction} & \to \{True, False\} \\ f_1, f_2 & \mapsto f_1 = f_2 \end{cases}$$

   ```
   type fraction = {num:int; denum:int};;
   let testEgalite1 f1 f2 = (f1.num * f2.denum - f2.num * f1.denum) = 0;;
   testEgalite1 {num = 1 ; denum = 2} {num = 3 ; denum = 6};;
   ```

2. Écrire la spécification et le code d'une fonction `testEgalite2` avec la notation sous forme de couple d'entiers.

Signature:

$$\begin{cases} \texttt{int}^2 \times \texttt{int}^2 & \to \{True, False\} \\ (a,b),(c,d) & \mapsto \frac{a}{b} = \frac{c}{d} \end{cases}$$

```
let testEgalite2 (a, b) (c, d) = (a * d - c * b) = 0;;
testEgalite2 (1, 2) (3, 6);;
```

N.B : On peut également vérifier que les fractions fournies en paramètre sont valides et gérer les cas d'erreur à l'aide d'un `failwith`.

Problème 2

1. Définir un type `couleur` où une couleur $c \in \{\text{Rose, Cyan, Violet, Orange, Rouge, Jaune, Vert, Bleu}\}$

```
type couleur = Rose | Cyan | Violet | Orange | Rouge | Jaune | Vert | Bleu;;
```

2. Définir un type `monopoly` comprenant un nom (`string`), une couleur (`couleur`) et un prix (`int`).

   Exemple :

```
# rue_1 :  monopoly = {nom = "Rue de la paix"; couleur = Bleu; prix = 450}
```

```
type monopoly = {nom:string; col:couleur; prix:int};;
```

3. Écrire la spécification et le code d'une fonction `sortRue` qui prend en entrée trois noms de rue et retourne ces trois rues (dans un triplet), ordonnées par ordre croissant de prix.

Voir fichier triMonop.ml sur Celene.