Durée du CC : 1h25

La Fiche-Triche Scala est le seul document autorisé.
Les deux parties sont indépendantes ; on pourra les traiter dans l'ordre de son choix.

1 Listes

Question 1 Implémentez une fonction

```
compterPM[T](element : T, liste: List[T]) : Int
```

qui renvoie le nombre de fois où `element` apparaît dans `liste`. Vous devrez utiliser un pattern matching au cœur de cette fonction.

Question 2 Implémentez une fonction

```
compterOL[T](element : T, liste: List[T]) : Int
```

qui renvoie le nombre de fois où `element` apparaît dans `liste`. Cette fonction devra tenir en une ligne (et ne devra donc pas utiliser de pattern matching).

2 Polynômes

Dans cette partie, nous allons représenter et manipuler des polynômes entiers à plusieurs variables. Un polynôme est constitué de sommes ($+$) et de produits ($\times$) d'entiers et de variables (qui seront des lettres).

Question 3 Déclarez une classe abstraite `Polynome` et scellez-là.
Déclarez ensuite les `case class` non-mutables suivantes, qui héritent de `Polynome` :

- `Entier(n)`, où `n` est un entier
- `Variable(c)`, où `c` est un caractère
- `Somme(p1,p2)`, où `p1` et `p2` sont des `Polynome`
- `Produit(p1,p2)`, où `p1` et `p2` sont des `Polynome`

Question 4 Implémentez la méthode `substituer()` de la classe `Polynome`. Celle-ci prendra en argument un `HashMap[Char,Int]` et renverra un `Polynome`.
Le résultat de `p.substituer(map)` est le `Polynome` `p`, où chaque `Variable(s)` a été remplacée par

- `Entier(v)` si le couple `(s,v)` est dans `map`
- `Variable(c)` sinon.

Autrement dit, cet appel substituera chaque variable par l'entier associé dans le `HashMap` passé en argument, en laissant inchangées les variables qui ne sont pas des clefs de ce `HashMap`.
Par exemple, si

```
val p = Produit(Somme(Variable('x'), Variable('x')), Variable('y'))
val map = HashMap('x' -> 4, 'z' -> 7)
```

alors `p.substituer(map)` renverra

```
Produit(Somme(Entier(4), Entier(4)), Variable('y'))
```

Question 5 Implémentez la méthode `evaluer()` de la classe `Polynome`. Celle-ci renverra une `Option[Int]`.
Le résultat de `p.evaluer()` sera de la forme `Some(res)` s'il n'y a pas de `Variable` dans `p` et que l'expression s'évalue en l'entier `res`. S'il y a une `Variable` dans `p`, alors la méthode renverra `None`.
Par exemple, si

```
val p = Produit(Somme(Entier(4), Entier(4)), Variable('y'))
val q = Produit(Somme(Entier(4), Entier(4)), Entier(3))
```

alors `p.evaluer()` renverra `None` et `q.evaluer()` renverra `Some(24)` (pusique $(4+4)\times 3 = 24$).

Question 6 (☠) Implémentez la méthode `equals` de la classe `Polynome` de sorte que deux `Polynome` soient considérés comme égaux s'ils représentent des polynômes *mathématiquement* égaux.
Par exemple,

```
Somme(Produit(Entier(2), Variable('x')), Variable('x'))
Produit(Variable('x'), Somme(Entier(1),Entier(2)))
```

sont égaux, car ils représentent tous les deux le polynôme $3x$.
Implémentez ensuite une fonction `hashCode()` intelligente (c'est-à-dire qui ne renvoie pas le même résultat pour tous les `Polynome`), compatible avec `equals()`.