Programmation fonctionnelle : feuille 4 2019-2020

Exercice 4.1 Dans un plan muni d'un repère orthonormé on associe au point M de coordonnées $(x\,,y)$ son affixe, le nombre complexe $z = x + i \cdot y$.
Pour représenter un point du plan, on utilisera donc le type

```
type mycomplex = C of float * float
```

1. Écrire l'accesseur `realpart` qui permet de récupérer la partie réelle d'un complexe.
2. Écrire l'accesseur `imagpart` qui permet de récupérer la partie imaginaire d'un complexe.
3. Définir une variable `c_origin` contenant le point de coordonnées $(0\,,0)$.
4. Définir une variable `p12` contenant le point $(1\,.\,,2\,.)$.
5. Implémenter les opérations
   `c_sum c1 c2`, `c_dif c1 c2`, `c_mul c1 c2`,
   `c_abs c`, `c_scal lambda c`, `c_exp c`
   qui permettent respectivement de calculer la valeur absolue d'un complexe, la somme, la différence, le produit de deux complexes, la multiplication d'un complexe par un scalaire (float), l'exponentielle complexe.

Exercice 4.2 Une transformation F du plan transforme chaque point M en son image M'.
Aux points M et M', on associe respectivement leurs affixes, z et z'. L'écriture complexe de la transformation F est $z' = f(z)$ où f est la fonction $C \to C$ qui à z associe z'.

1. L'écriture complexe d'une translation de vecteur $\overrightarrow{u}$ est $z' = z + b$ où b est l'affixe du vecteur $\overrightarrow{u}$.
2. L'écriture complexe d'une rotation de centre Ω d'affixe ω et d'angle θ est $z' = e^{i\cdot\theta} \cdot (z - \omega) + \omega$.
3. L'écriture complexe d'une homothétie de centre Ω d'affixe ω et de rapport k est $z' = k \cdot (z - \omega) + \omega$

Implémenter les transformations translation, rotation et homothétie d'un point M en son image M' en utilisant leurs écritures complexes données ci-dessus et les opérations sur les complexes écrites dans l'exercice précedent.

Exercice 4.3 Dans leur article "Haskell vs. Ada vs. C++ vs. Awk vs. ... An Experiment in Software Prototyping Productivity http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.368.1058&rep=rep1&type=pdf de 1994, Paul Hudak et Mark P. Jones rapportent une expérience rare de comparaison entre différents langages de programmation. Lelogiciel implémenté manipule des zones géométriques. Nous allons étudier une version simplifiée (mais pas tant que ça) de ce logiciel.

Une zone géométrique est un ensemble de points du plan. On représente une telle zone par sa *fonction caractéristique*, c'est-à-dire par la fonction booléenne qui prend un point en argument, et retourne `true` si le `point` appartient à la zone et `false` sinon. Ainsi, une zone vide est représentée par la variable `nowhere` suivante :

```
# let nowhere = fun point -> false;;
val nowhere : 'a -> bool = <fun>
```

1. Définir une variable `everywhere` représentant la zone correspondant au plan tout entier.
2. Écrire une fonction `point_in_zone_p` telle que `point_in_zone_p point zone` retourne `true` si le

point `point` appartient à la zone `zone`.

Exemples :

```
# point_in_zone_p c_origin nowhere;;
- : bool = false
# point_in_zone_p c_origin everywhere;;
- : bool = true
```

Exercice 4.4 1. Écrire une fonction `make_rectangle width height` qui retourne la zone rectangulaire définie par les points $(0,0)$, $(width,0)$, $(width,height)$, $(0,height)$, c'est-à-dire la fonction qui prend un point en argument et retourne `true` si cet argument est dans le rectangle (ou sur sa bordure), et `false` sinon.

2. Écrire une fonction `zone_union zone1 zone2` qui retourne l'union des zones `zone1` et `zone2` (c'est-à-dire l'ensemble des points qui appartiennent à l'une ou l'autre des zones).
3. Écrire une fonction `zone_complement zone1` qui retourne la zone correspondant aux points ne se trouvant pas dans la zone `zone1`.
4. Écrire une fonction `zone_intersection` telle que `zone_intersection zone1 zone2` retourne la zone correspondant aux points se trouvant à la fois dans les zones `zone1` et `zone2`.

Exercice 4.5 1. Écrire une fonction `make_disk0 radius` qui retourne un disque de rayon `radius` centré au point $(0,0)$.

2. Dans un repère orthonormé, dessiner la zone définie par l'expression suivante.
   `zone_union (make_rectangle 2. 4.) (make_disk0 3.)`

Exercice 4.6 S'inspirer de la fonction `move_zone` pour rajouter une fonction `scale_zone0`. Cette nouvelle fonction aura deux paramètres, le premier est la zone à transformer, et le deuxième est le point (λ_1, λ_2). La fonction applique à la zone la transformation

$$(x, y) \mapsto (\lambda_1 x, \lambda_2 y).$$

Écrire une fonction `scale_zone zone coeff point` qui effectue la mise à l'échelle par rapport au point `point` et non pas par rapport à l'origine.

Exercice 4.7 Écrire une fonction `make_disk radius point` qui retourne un disque de rayon `radius` centré au point `point`. `p`.

Exercice 4.8

```
let test =
  let c = make_disk0 1. in
  let c1 = move_zone c (C(1., 0.)) in
  assert (point_in_zone_p (C(0.0, 0.5)) c);
  assert (not (point_in_zone_p (C(1.0, 0.5)) c));
  assert (point_in_zone_p (C(0.5, 0.)) (zone_intersection c c1));;
```

En vous inspirant du jeu de tests ci-dessus, écrire un jeu de test pour toutes les fonctions.

Exercice 4.9 1. Vérifier la présence des fichiers `images.cmo` et `visu-zones.ml`.

2. Regarder le contenu du fichier `visu-zones.ml`.
3. Compiler le fichier `visu-zones.ml`.
4. Utiliser la fonction `view_zone zone` pour produire une image PNG et visualiser une partie finie d'une zone.
5. Pour adapter la taille de la zone visualisée, utiliser `view_zone zone size`.

Exercice 4.10 1. Écrire une fonction `rotate_zone0 zone angle` qui effectue une rotation d'angle `angle` autour de l'origine.

2. Écrire une fonction `rotate_zone zone angle point` qui effectue une rotation d'angle `angle` autour du point `point`.