info421 : Programmation fonctionnelle
TD 7 : références, mémoization

Responsables : Pierre Hyvernat et Krzysztof Worytkiewicz
Laboratoire de mathématiques de l'université de Savoie
email : Pierre.Hyvernat@... / Krzysztof.Worytkiewicz@... (...@univ-savoie.fr)
http://lama.univ-savoie.fr/~hyvernat/
http://lama.univ-savoie.fr/~worytkiewicz/

Exercice 1 : références

Question 1. Quels sont les types, et les valeurs de définitions suivantes ?

```
let e1 = [];;
let e2 = ref [];;
let e3 =
  let r = ref 1
  in !r+1;;
let e4 =
  let r = ref 2
  in r := !r+1;;
let e5 =
  let r = ref (-1)
  in r := !r-1;
     !r-1;;
```

Question 2. Dans un langage impératif, une variable globale est accessible en lecture et écriture par toutes les fonctions du programme.

En Caml, une définition

```
let u = 42;;
...
```

est accessible en lecture par toutes les fonctions du fichier. Par contre, `u` est une constante et n'est pas modifiable.

Comment peut-on créer une *variable* globale en Caml ?

Question 3. Donnez un exemple de programme Caml avec des références qui ne satisfait pas la transparence référentielle.

Question 4. Imaginez que vous avez un gros programme comportant de nombreuses fonctions. Une de ces fonctions, la fonction `f`, vous intéresse particulièrement, et vous aimeriez savoir combien de fois vous l'appelez lors d'une exécution de votre programme.

Expliquez comment procéder en utilisant une référence.

Exercice 2 : Mémoization

Question 1. Vous aimeriez optimiser le calcul des valeurs d'une fonction `f` de la manière suivante : lors du calcul de `f x`, vous regardez si la valeur correspondante a déjà été calculée.

- Si oui, vous renvoyez la valeur déjà calculée
- si non, vous calculez `f x` normalement, et conservez la valeur quelque part.

(Cette technique s'appelle "mémoization". Le deuxième calcul de `f x` est en général plus rapide que le premier...)

Question 2. Que pensez-vous de la complexité de `memoization f` par rapport à celle de `f` ?

Question 3. Est-ce que la transparence référentielle est conservée ?

Question 4. Peut-on mémoizer une fonction qui ne respecte pas la transparence référentielle. Donnez un exemple.

Question 5. En utilisant les remarques des questions précédentes, écrivez une fonction `mémoization : ('a->'b) -> ('a->'b)` qui transforme une fonction (presque) quelconque en une version mémoizée.

Question 6. Expliquez comment vous pourriez améliorer encore un peu la complexité de votre fonction...

Question 7. Écrivez une fonction mémoizée `fib : int -> int` pour calculer la fonction de Fibonacci.

Peut-on écrire une version générale de `memoization` qui fonctionne correctement pour les fonctions récursives ?