Informatique *TP*
Logique Ipesup

1 Exercice

Proposer une implémentation OCaml de tableaux redimensionnables *polymorphes*, c'est-à-dire un type `'a vector`.

2 Exercice

Modifier la fonction `vector_resize` pour qu'elle rétrécisse le tableau interne au tableau redimensionnable dès que

$$\texttt{size} < \texttt{capacity}/4.$$

3 Exercice

Écrire une fonction

```
length : 'a list -> int
```

qui calcule la longueur d'une liste sans risquer de faire déborder la pile d'appels.

Indication : commencer par écrire une fonction plus générale qui calcule la somme d'un entier et de la longueur d'une liste.

4 Exercice

Écrire une fonction qui reçoit en argument une liste non vide et qui renvoie un élément aléatoire de cette liste, avec équiprobabilité. Essayer de le faire en parcourant la liste *une seule fois*.

5 Exercice

Écrire une fonction

```
list_of_array : 'a array -> 'a list
```

qui transforme un tableau en liste. On veillera à ne pas faire déborder la pile d'appels.

6 Exercice

Écrire une fonction

```
array_of_list : 'a list -> 'a array
```

qui transforme une liste en tableau. On veillera à ne pas faire déborder la pile d'appels.

7 Exercice

Écrire une fonction C

```
list *list_interval(int lo, int hi)
```

qui renvoie la liste constituée des entiers

$$lo,\ lo+1,\ \ldots,\ hi-1,$$

dans cet ordre, et la liste vide si $hi \leq lo$. L'écrire avec une boucle `while`.

8 Exercice

Les listes chaînées étant mutables, rien ne nous empêche de modifier un champ `next` pour le faire revenir sur un élément précédent de la liste et créer ainsi une liste cyclique à partir d'un certain rang.

On peut se poser alors la question d'écrire un programme pour déterminer si une liste donnée est ou non cyclique à partir d'un certain rang, c'est-à-dire une fonction

```
bool list_cyclic(list *l).
```

Pour cela, on pourrait parcourir la liste et stocker dans une structure de données tous les pointeurs rencontrés, en vérifiant à chaque nouvelle cellule de la liste qu'elle n'a pas déjà été vue. Ce serait linéaire en temps et en espace si par exemple on utilisait une table de hachage pour stocker les pointeurs.

On peut cependant faire beaucoup mieux, en utilisant l'algorithme du *lièvre et de la tortue* inventé par Robert Floyd. Il consiste à parcourir la liste avec deux pointeurs, l'un partant du premier élément et avançant à la vitesse un (la tortue) et l'autre partant du deuxième élément et avançant à la vitesse deux (le lièvre).

Si le lièvre parvient au bout de la liste (`NULL`), alors il n'y a pas de cycle. Sinon, les deux pointeurs finiront nécessairement par être égaux et la liste est cyclique à partir d'un certain rang.

1. Justifier la correction de cet algorithme.
2. Montrer qu'il termine toujours.
3. Écrire une fonction C réalisant cet algorithme.