Question 1 On souhaite représenter et manipuler en OCaml les directives utilisées dans les descriptions de pistes de carte au trésor. Par exemple : Avancer de 4 pas, tourner à gauche ; reculer de 10 pas, etc. On suppose que les directives simples sont : Avancer de n pas en avant, Reculer de n pas en arrière, Tourner à gauche, Tourner à droite. On définie donc les types directive et path pour représenter la description d'une piste :

```
type directive = TurnLeft | TurnRight
      | StepForward of int
      | StepBackward of int;;
type path = directive list;;
let sample_path = (StepForward 1 :: StepForward 2:: TurnLeft ::
  StepBackward 3 :: TurnLeft :: StepForward 1:: []);;
```

1. Définir la fonction `inverse : directive -> directive` retournant l'inverse d'une directive ; (l'inverse de tourner a droite est tourner à gauche, l'inverse d'avancer est reculer).
2. Définir la fonction `string_of_path : path -> string` qui retourne une chaîne de caractères décrivant les étapes successives d'une piste ; par exemple `string_of_path sample_path` devrait retourner "Avancer de 1 pas ; Avancer de 2 pas ; Tourner à gauche ; Reculer de 3 pas ; Tourner à gauche ; Avancer de 2 pas". On notera l'absence de ; à la fin de la chaîne !
3. Définir la fonction simplify : path -> path simplifiant une piste en supprimant toutes les directives successives inverses (cela correspond à éviter de piétiner). Attention ce n'est pas facile : `simplify [TurnLeft; StepForward 2; StepBackward 2; TurnRight ]` renvoi `[]` par contre `simplify [StepForward 2; StepBackward 3]` ne se simplifie pas.

Question 2 L'état du chasseur de trésor est donné par un triplet `(x, y, o)` représentant ses coordonnées en 2D et son orientation (nord, sud, est, ouest).

1. Définir le type orientation des 4 orientations : nord, sud, est, ouest (mais en anglais s'il vous plaît) ;
2. Définir le type hunter permettant de représenter l'état d'un chasseur ;
3. Définir la fonction `string_of_hunter : hunter -> string` qui renvoit une chaîne de caractère representant le chasseur. Tel que `string_of_hunter (3,2,South)` renvoie `"(3,2,sud)"` ;
4. Définir la fonction `move: hunter -> directive -> hunter` calculant le nouvel état d'un chasseur après avoir effectué une directive ;
5. Définir la fonction `finally: hunter -> path -> hunter` calculant l'état final après avoir suivi une piste à partir d'un état initiale ; Faire deux versions : une version récursive et une version utilisant la fonction `List.fold_left` déjà présente dans OCaml (qui ressemble beaucoup à votre fonction `combine_all`).

Question 3 On souhaite maintenant introduire des obstacles sur la carte. On décrira les obstacles avec des listes de positions. Si une position est sur cette liste, alors un obstacle est sur cette position, et la position est donc inaccessible.

```
type obstacles = (int * int) list;;
```

1. Définir la fonction `move_with_obstacles : obstacles -> hunter -> directive -> hunter` calculant le nouvel état d'un chasseur après avoir effectué une directive. Si un obstacle se trouve sur le chemin, le chasseur avancera jusqu'au premier obstacle et s'arrêtera ; utiliser votre fonction `contains` du TP 06 (qui existe d'ailleurs déjà dans OCaml sous le nom `List.mem`) ;

2. Définir la fonction `finally_with_obstacles: obstacles -> hunter -> path -> hunter` calculant l'état final après avoir suivi une piste à partir d'un état initiale en utilisant `List.fold_left`.

Question 4

1. Créer un répertoire vide «TP08», et y créer deux modules : un fichier path.ml contenant les réponses de l'exercice 1, et un fichier hunter.ml contenant les réponses des exercices 2 et 3. Ajouter au début du fichier hunter.ml la directive open Path ;;. Ce découpage n'est pas forcément idéal, il n'est là que pour nous faire la main.
2. Compiler vos modules en faisant :

```
ocamlc -c path.ml
ocamlc -c hunter.ml
```

3. Tester vos modules en faisant :

```
ocaml path.cmo hunter.cmo
# open Path;;
# open Hunter;;
# (* ici refaites vos tests *)
```