CSE 341, Fall 2004, Assignment 2
Due: Monday 18 October, 9:00AM

Version 1

You will write 10 SML functions having to do with "Tetris moves" and "Tetris pieces". Your solutions must use pattern-matching. You may not use the functions `null`, `hd`, or `tl`, nor may you use anything containing a `#` character. You may not use mutation. You *may* use ML's built-in append operator (`@`). Style matters. The sample solution is roughly 115 lines.

All necessary Tetris knowledge is here; ask if something is unclear.
A player can *move* a piece left or right or rotate it clockwise or counterclockwise:

```
datatype player_move = Left | Right | Clockwise | CounterClockwise
```

Given a list of moves, a piece moves a *distance* and a *rotation*. The distance is "number of `Right` moves minus number of `Left` moves" (and can be negative). For rotation, a piece starts at "0 degrees", `Clockwise` "substracts 90 degrees" and `Counterclockwise` "adds 90 degrees", but as usual, the rotation is is always between 0 and 360. We can *summarize* the effect of a list of moves with these types:

```
datatype rotation = R0 | R90 | R180 | R270
type move_summary = {distance : int, rotation : rotation}
```

Squares and *pieces* are represented like in homework 1. A *piece-maker* is a `next_square list`:

```
datatype next_square = North | South | East | West
```

The piece that a piece-maker `p` makes depends on a *current-square* `(x,y)` and is defined as follows:

- If `p` is `[]`, the piece contains 1 square, which is `(x,y)`.
- If `p` is `North::p2`, then the piece contains `(x,y)` and the piece made by `p2` with current-square `(x,y+1)`.
- Similarly, `South` changes the current-square "down 1", `East` "right 1", and `West` "left 1".

1. (Summarizing Moves) Write these functions:
   (a) `change_distance` takes a `player_move` m and an `int` d and evaluates to the distance a piece would travel if we did the move m after the piece had traveled distance d. (Hint: This function is easier to write than describe. Some moves do not change the distance traveled.)
   (b) `rotate` takes a `player_move` m and a rotation r and evaluates to the rotation the piece would turn to if we did the move m when the piece was already at rotation r. (Hint: For an elegant solution, pattern-match on the pair (m, r).)
   (c) `summarize_move` takes a `player_move list` and evaluates to the `move_summary` describing the distance and rotation of a piece after the moves, assuming the piece started at distance 0 and rotation 0 degrees. (Hint: The order of the moves does not matter. Use earlier functions.)

2. (Unsummarizing Moves) Write these functions:
   (a) `make_n_moves` takes a `player_move` m and an `int` n and evaluates to to a list with n moves m.
   (b) `unsummarize_move` takes a `move_summary` and evaluates to a minimal-length `player_move list` that the summary correctly summarizes. (Hint: Use `make_n_moves`.)

3. (Generating Piece-Makers) Write these functions:
   (a) `add_to_all` takes a `next_square` n and a `next_square list list` *lst* and evaluates to a list where the i^{th} element is n consed onto the i^{th} element of *lst*. (Hint: If you do not give explicit types, `add_to_all` will have type `'a * 'a list list -> 'a list list`; this is fine.)

(b) `all_next_squares` takes a number n and returns a `next_square list list` that has every length-n `next_square_list` in it exactly once and no other elements. (Hint: Use `add_to_all`. The length of `all_next_squares` n is 4^n (so don't pass large numbers). $4^0 = 1$.)

4. (Filtering Repeated-Square Makers) Write these functions

   (a) `make_piece` takes a `next_square list` *lst* and returns the piece obtained from starting at `(1,1)` and following the directions in *lst*. (Hint: Use a helper function that takes a list, and current x and y coordinates.)

   (b) `has_repeat` takes a piece (type `(int*int) list`) and evaluates to true if two squares in the piece are the same.

   (c) `filter_repeats` takes a `next_square list list` and evaluates to a `next_square list list`. The result list is a subset of the argument list; it containst exactly those elements that make pieces *without* repeated squares. (Hint: Use earlier functions.)

5. (**Extra Credit:** Tail-Recursive Piece-Generation) Write `all_next_squares2`, which given the same argument as `all_next_squares` evaluates to the same result. However, `all_next_squares2` must call only a tail-recursive helper function. This helper function must be tail recursive and call only itself and built-in operations (such as `-` and `::`). Hint: Sample solution is 11 lines.

Type Summary: Evaluating a correct solution should generate these bindings (allowing `move_summary` to replace `{distance:int, rotation:rotation}` and vice-versa because they are type synonyms):

```
datatype player_move = Clockwise | CounterClockwise | Left | Right
datatype rotation = R0 | R180 | R270 | R90
type move_summary = {distance:int, rotation:rotation}
datatype next_square = East | North | South | West

val change_distance = fn : player_move * int -> int
val rotate = fn : player_move * rotation -> rotation
val summarize_move = fn : player_move list -> {distance:int, rotation:rotation}
val make_n_moves = fn : player_move * int -> player_move list
val unsummarize_move = fn : move_summary -> player_move list
val add_to_all = fn : next_square * next_square list list -> next_square list list
val all_next_squares = fn : int -> next_square list list
val make_piece = fn : next_square list -> (int * int) list
val has_repeat = fn : (int * int) list -> bool
val filter_repeats = fn : next_square list list -> next_square list list
```

Of course, generating these bindings does not guarantee that your solutions are correct: *Test your functions.*

Turn-in Instructions

- Put all your solutions in one file, `lastname_hw2.sml`, where `lastname` is replaced with your last name.
- The first line of your `.sml` file should be an ML comment with your name and the phrase `homework 2`.
- Email your solution to `brianhk@cs.washington.edu`.
- The subject of your email should be *exactly* `[cse341-hw2]`.
- Your `.sml` file should be an *attachment*.