CSE341 Spring '05 Due Friday, April 15
Assignment 2

For this homework your solutions must use pattern-matching. You may not use the functions `hd`, `tl`, `null` or anything containing the `#` character. Similarly, don't use if-then-else in places where pattern matching will suffice (although there are a small number of places where you will need if-the-else).

1. More Battleship! We'll formalize the way we represented a ship in the previous homework by making a `grid_ship` type:

```
type grid_ship = (int*int) list
```

While this is useful for determining if shots hit, it isn't so good at placing a ship—for example, there's no guarantee that the squares composing a ship are are all adjacent. So we define the following type.

```
datatype orientation = Horiz | Vert
type ship = { loc : (int*int), dir : orientation, size : int }
```

Here `loc` is the corner closest to `(0,0)` (the upper-left corner of a regulation battleship board, note this is different from mathematics).

   (a) Define functions `make_horiz` and `make_vert` that take an `int`, and return a horizontal or vertical `grid_ship`, respectively, of the specified length, starting at `(0,0)` (note that ML's type inference may come up with `(int*int) list` instead of `grid_ship` and vice-versa). Hint: recursion and pattern-matching make each of these functions 2 lines apiece. For example, `make_horiz 3` evaluates to something like `[(0,0),(1,0),(2,0)]` and `make_vert 2` to `[(0,0),(0,1)]`. The order of items in the list doesn't matter.

   (b) Define a function `shift` that takes a `grid_ship` and an `int*int` pair and returns a `grid_ship` shifted by the specified amount. So `shift([(1,2),(1,3)],(2,3))` becomes `[(3,5),(3,6)]`.

   (c) Define a function `ship_to_grid` that takes a `ship` and returns `grid_ship`. Hint: use the previous functions. `ship_to_grid{loc=(3,1), dir=Horiz, size=4}` would evaluate to `[(3,1),(4,1),(5,1),(6,1)]`.

   (d) Define a function `fleet_to_grid` that takes a `ship list` and returns `grid_ship list`: a list version of the previous function.

   (e) Define a type `shot_summary` that records the number of hits and misses on a fleet. Use a record type defined as implied by the bindings below.

   (f) Define a function `fleet_salvo_summary` that takes a list of shots (a salvo) and a `grid_ship list` and returns a `shot_summary` that summarizes the number of hits and misses the fleet has sustained. You may assume that no ships overlap and all the shots are distinct. Modify the `shot_hit` function from last homework to use pattern matching and use it (also include it in your solution this week). It may be helpful to define some local functions in `fleet_salvo_summary`. This function is similar to the `salvo_hits` function from last week.

2. Consider the following type definitions.

```
datatype bit = One | Zero
type binary_number = bit list
```

Write a function `eval_bin` that evaluates a `binary_number`. For example, `eval_bin([One,Zero,Zero])=>4:int` (note it does *not* evaluate to 1!). You *must* write this function in an accumulator style: you must have a local function `eval_inner` of type `binary_number*int->int` that is tail recursive (this is actually the natural way to solve this problem, anyway...).

3. In the following we will represent sets as lists.

   (a) Consider the following.

   ```
   infix mem
   fun x mem [] = false
     | x mem (y::ys) = x=y orelse x mem ys
   fun newmem(x,xs) = if (x mem xs) then xs else x::xs
   ```

   Type these up and include them in your turn-in. In a comment near these functions, answer the following three questions. What is the result of `newmem(2,[1,2])`? Of `newmem("apple" ,["orange","banana"])`? Describe in your own words what these functions do, using only one sentence for each function.

   (b) Write a function `setof` that takes a list of items, possibly with duplicates, and returns a list with all the duplicates removed. For example, `setof [1,2,3,2] =>[1,2,3]`. Hint: use `newmem`. The order of items in the output list doesn't matter.

   (c) Write an infix function `union` that computes the union of two lists of items when viewed as sets. `[1,2,3] union [2,3,4]` evaluates to `[1,2,3,4]` (or the same list in a different order).

   (d) Write an infix function `isect` that computes the intersection of two lists of items when viewed as sets, evaluating `[1,2,3] isect [2,3,4]` as `[2,3]` (again, the order may be different).

   (e) What is the main advantage of not declaring the argument types of these functions? Include the answer as a comment after your definition of `isect`.

   (f) The *Cartesian product* of two sets $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_m\}$ is the set of all pairs $\{(a_i, b_j)\}_{i \le n, j \le m}$. Write an accumulator function `one_cross` that forms the cross product of a single element with a set, so that `one_cross(1, [1,2,3], nil)` evaluates to `[(1,1),(1,2),(1,3)]` (as usual, the order may be different).

   (g) Write `one_cross_noaccum` that is the same as above, but not written in accumulator style (and not using the above `one_cross`). In a comment near the function state why this function is not tail recursive.

   (h) Write an infix function `cross` that takes two lists and returns their Cartesian product. You may assume the lists are already sets, and you must use the accumulator version `one_cross`. You may use the append operator "`@`", but try to do it without append.

Type Summary: A correct assignment will generate the following bindings. That means a solution that differs from these bindings (except by type synonyms like `grid_ship` vs. `(int*int) list`) is incorrect. My solution is 80 lines.

```
type grid_ship = (int * int) list
datatype orientation = Horiz | Vert
type ship = {dir:orientation, loc:int * int, size:int}
val make_horiz = fn : int -> (int * int) list
val make_vert = fn : int -> (int * int) list
val shift = fn : (int * int) list * (int * int) -> (int * int) list
val ship_to_grid = fn
  : {dir:orientation, loc:int * int, size:int} -> (int * int) list
val fleet_to_grid = fn
  : {dir:orientation, loc:int * int, size:int} list -> (int * int) list list
type shot_summary = {hits:int, misses:int}
val shot_hit = fn : (''a * ''b) * (''a * ''b) list -> bool
val fleet_salvo_summary = fn
  : (''a * ''b) list * (''a * ''b) list list -> {hits:int, misses:int}
datatype bit = One | Zero
type binary_number = bit list
val eval_bin = fn : bit list -> int
infix mem union isect cross
val mem = fn : ''a * ''a list -> bool
val newmem = fn : ''a * ''a list -> ''a list
val setof = fn : ''a list -> ''a list
val union = fn : ''a list * ''a list -> ''a list
val isect = fn : ''a list * ''a list -> ''a list
val one_cross = fn : 'a * 'b list * ('a * 'b) list -> ('a * 'b) list
val one_cross_noaccum = fn : 'a * 'b list -> ('a * 'b) list
val cross = fn : 'a list * 'b list -> ('a * 'b) list
```