Suggested Solutions to Midterm Problems: Part II

1. (5 points) What is the response of the OCaml interpreter to the following expression? Please explain what the response means.

```
let rec remove_if f = function
    [] -> []
  | a::y -> if f a then remove_if f y else a :: remove_if f y
```

Solution. The response is as follows:

```
val remove_if : ('a -> bool) -> 'a list -> 'a list = <fun>
```

It means that `remove_if` is a polymorphic function that takes as inputs a function of type t to `bool` and a list with elements of type t and outputs a list with elements also of type t, where t can be any type. It also means that `remove_if`, after taking a function of type t to `bool` as the only input, will output a function of type t `list` to t `list`, where t again can be any type. □

2. (15 points)

   (a) Declare a datatype in OCaml for representing binary search trees that store integral key values.

   Solution.

```
type stree =
    Nil
  | Node of stree*int*stree
```

□

   (b) Define a function that, given an integer and a binary search tree as inputs, determines whether the integer is in the tree.

   Solution.

```
let rec find n t =
  match t with
    Nil -> false
  | Node (t1,m,t2) ->
      if m=n then true
      else if m>n then find n t1
      else find n t2
```

□

(c) Define a function that inserts an integer into a binary search tree.

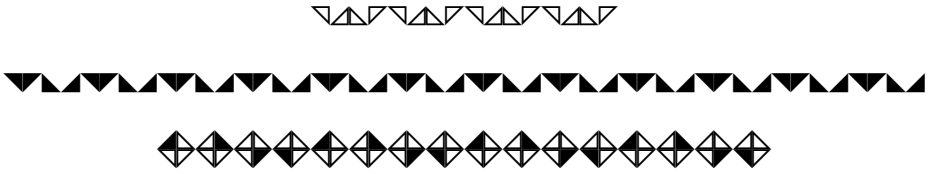
Solution.

```
let rec insert n t =
  match t with
    Nil -> Node (Nil,n,Nil)
  | Node (t1,m,t2) ->
      if m=n then t
      else if m>n then Node(insert n t1,m,t2)
      else Node(t1,m,insert n t2)
```

□

3. (20 points) This problem builds on our implementation of Little Quilt in OCaml, which was discussed in class.

(a) Define suitable auxiliary functions so that the following quilts can be easily defined and shown (printed).



Explain by examples how the auxiliary functions may be used.

Solution.

```
let rec copy n f s =
  if n = 1 then s
  else sew s (copy (n-1) f (f s))
```

The three example quilts in the problem statement may be produced respectively by the following three expressions.

```
show (copy 16 turn a);;
show (copy 48 unturn b);;
let diamond = pile (sew (turn b) (turn (turn a))) (sew a (unturn a)) in
show (copy 16 turn diamond);;
```

□

(b) Suppose we wish to include the following two objects as primitive square pieces:

Modify the OCaml implementation of Little Quilt so that one can define and show quilts which may contain the additional square pieces.

Solution. Let us call the two new primitive pieces (when treated as quilts) `c` and `d` respectively. The needed changes are as follows.

```
type texture = WTriangle | BTriangle | WSquare | BSquare

let sqc = (WSquare,NE)
let sqd = (BSquare,NE)
let c = [[sqc]]
let d = [[sqd]]

let encode = function
    (WTriangle,NE) -> "◹"
  | (WTriangle,SE) -> "◿"
  | (WTriangle,SW) -> "◺"
  | (WTriangle,NW) -> "◸"
  | (BTriangle,NE) -> "◥"
  | (BTriangle,SE) -> "◢"
  | (BTriangle,SW) -> "◣"
  | (BTriangle,NW) -> "◤"
  | (WSquare,_) -> "□"
  | (BSquare,_) -> "■"
```

□

4. (10 points) Implement a Common Lisp function that takes two lists A and B (as sets) and computes a list representing $A \setminus B$ (the set difference between A and B).

   Solution. (Jui-Shun Lai)

```
(defun isMember (e s)
  (cond ((null s) nil)
        ((= e (car s)) t)
        (t (isMember e (cdr s)))))

(defun difference (s1 s2)
  (cond ((null s1) nil)
        ((isMember (car s1) s2) (difference (cdr s1) s2))
        (t (cons (car s1) (difference (cdr s1) s2)))))
```

□