CSci 4651 Spring 2006
Problem Set 4: Imperative programming, ML
Due Friday, March 17

Some notes on running ML. ML code may be written in a separate file and then loaded into ML run-time system using command `#use`. For instance,

```
#use ``myfile.sml'';;
```

loads the file `myfile.sml` in the current directory into the ML run-time system. All the function definitions from that file become available to ML system. The response to a file load is a sequence of responses to the definitions in the file. See the link to the description of ML toplevel system from the resources page for more discussion on file loading.

Keep in mind that `#quit;;` quits the interpreter.

Problem 1. Finish the lab problems (see the web page).
Problem 2. The function `traverse` defined below works exactly as the function `traverse` in the assignment on Scheme.

```
let rec traverse combine action seed = function
  | [] -> seed
  | x :: xs -> combine (action x) (traverse combine action seed (xs)) ;;
```

Below is an example of using `traverse`:

```
let comb x y = x :: y;;
let mult x = x * x;;
let mapsquare = traverse comb mult [];;
mapsquare [1; 5; 7; -2];;
```

Note that, unlike in Scheme, in ML `*` is an operator, not a function. In other words, `*` makes sense only between two numbers, not by itself, so you cannot pass it to a function. That's why the second parameter to `traverse` in the example above is `mult x = x * x`, and not just `*`. The same holds for `+`, `-`, and other operators and for `::`.

The next example shows how to use traverse to define a function `count` to count the number of elements in the array:

```
let comb x y = x + y;;
let act x = 1;;
let count x = (traverse comb act 0) x;;
```

Note that removing the `x` on both sides will produce a function that could only be applied to one type after it is defined, not to any type.
Define the following functions using `traverse`:

1. `sumlist` to add up all the elements of an integer list.

2. `remove5` to remove all 5s from a list of integers.

3. `min` to find a minimum element in a list of integers Make an assumption about the largest number that may appear on a list.

4. `reverse` to reverse a list. You may use `append` from the lab for this question.

You may define auxiliary functions if you find them helpful. Note that "do" is a reserved word in ML which can't be used as a parameter name. That's why the second parameter of `traverse` is renamed to `action`.

Problem 3. You are given the following definition of a tree type (you can copy/paste it from the file with the strating OCaml code, the file also has a couple of sample trees):

```
type 'a tree = Empty | Node of 'a * 'a tree * 'a tree;;

let intTree = Node (5, Node (3, Empty, Empty),
Node (6, Node (7, Empty, Empty), Empty));;

let strTree = Node ("apples", Empty, Node ("bananas",
Node ("oranges", Empty, Node ("grapes", Empty, Empty)), Empty));;
```

Question 1. Draw the pictures of the two given trees.

Question 2. Write a function `traversetree` which takes three parameters, `combine`, `action`, and `seed`, and returns a function that works on trees, much like traverse works on lists. Here are more details:

- `combine` is a function of three arguments since it combines the result of `action` on the value of the node with the results of the recursive calls on the two subtrees. Notice that you have to be consistent with how you use `combine` and how you define the function that you pass as a value for `combine`: if `combine` is used as

  ```
  combine x left right
  ```

  (i.e. so that it takes the three paaremeters separately), the function that you pass for combine should be defined with three separate parameters. Alternatively you can use `combine` as

  ```
  combine (x, left, right)
  ```

  (i.e. so that it takes a triple), and cosequently any function passed for `combine` must take a triple.

- `action` is applied to the value of the node.

- `seed` is the value returned from an empty tree.

Use `traversetree` to generate the following functions:

- `addtree` to add all elements of a tree of integers,
- `concattree` to concatenate all strings in a tree of strings. ^ is the string concatenation operation.
- `flattentree` which returns all the elements of the tree as a list. For instance, for `intTree` above it should return `[5; 3; 6; 7]`. The order of elements in the list doesn't matter. You may use @ for appending two lists. **Important:** your function must work for lists of any type. See the `count` example in Problem 2 for the correct syntax.
- **Extra credit.** Using `traversetree`, construct a function `find` that takes two arguments, which are a tree and an element, and returns true if the element is in the tree and false otherwise. Hint: tt action is the one that does the comparison.

  Note: this is a difficult problem, don't spend a lot of time on it if you have more important things to do, like Sem II or other things needed for graduation.