Assignment 2: Sets and Modules
CS 131, Spring 2001

Out: Wednesday, January 31

Due: Wednesday, February 7, 11:00am

A small part of the assignment has been done for you, so start by retrieving and reading the file `assign2.sml` from the course web pages. Complete the `assign2.sml` file and turn it in using `cs131submit`. Remember that your submitted files *must* compile without errors; code that cannot be compiled must be commented out.

The problems all make use of the following definitions:

```
signature SET = sig
                  type item
                  type set
                  val empty   : set                (* Empty set *)
                  val member  : set * item -> bool (* Check membership *)
                  val add     : set * item -> set  (* Create larger set *)
                end

signature INTSET = SET where type item = int
```

1 Sets as Lists

Implement a structure `ListSet` satisfying the `INTSET` signature that implements sets as lists of integers. Your functions must maintain the representation invariant that an integer never appears in a list more than once.

For debugging purposes it may be helpful to comment out the `:> INTSET` signature ascription. Just remember to uncomment this before handing in your code (and make sure the code still typechecks).

2 Sets as Functions

Implement a structure `FunctionSet` satisfying the `INTSET` signature that implements sets as functions from integers to booleans (returning true or false according to whether the given argument is in the set or not).

3 Sets as Binary Trees

Implement a structure `TreeSet` satsifying the `INTSET` signature that implements sets as ordered binary trees. You should use the following datatype to represent trees:

```
datatype tree = Leaf
              | Node of tree * int * tree
```

(This datatype is similar to examples you saw in the second lecture, except that there are integers on the interior nodes but none at the leaves.) Recall that the invariant for an ordered binary tree is that at each node the integers in the left subtree are less than the integer in the node, which is in turn less than all the integers in the right subtree.

Remember that the function `add` does not change the set given as an argument, but rather returns a "new" set; hence the `add` function will have to create new tree nodes as necessary.