1. Solve Exercise 3.2 in page Lecture 3.

2. Consider the following skeleton of a functional module for (binary) trees and (non-empty) lists of natural numbers. For your convenience, it imports the module NAT in the Maude prelude, which has all the usual numerical functions and predicates on numbers that you might need, as well as the Booleans and if_then_else-fi that are also available.

```maude
fmod LIST+TREE is protecting NAT .
sorts NeList Tree .
subsort Nat < NeList .
subsort Nat < Tree .
op _#_ : Tree Tree -> Tree [ctor] .

vars N M : Nat . vars T T1 T2 : Tree . vars L L1 L2 : NeList .
op _@_ : NeList NeList -> NeList . *** list append
*** include your equations defining _@_ here
op rev : NeList -> NeList . *** list reverse
*** include your equations defining rev here
op sort : NeList -> NeList . *** list sort
*** include your equations defining sort here
op add : NeList -> Nat . *** adds numbers in list
*** include your equations defining add here
op trev : Tree -> Tree . *** tree reverse
*** include your equations defining trev here
op add : Tree -> Nat . *** adds numbers in tree
*** include your equations for add here
op t2l : Tree -> NeList . *** converts tree into list
*** include your equations defining t2l here
```
Note that natural numbers are a subsort of both non-empty lists and binary trees. Except for the imported module NAT, only the data, that is, the constructor terms, of this module are defined. What you are asked to do is to define the following functions on such data:

- \texttt{__@__} appends to non-empty lists
- \texttt{rev} reverses a non-empty list
- \texttt{sort} sorts a non-empty list
- \texttt{add} adds all the numbers in a non-empty list
- \texttt{trev} reverses a binary tree with numbers as leaves; geometrically, it returns the \textit{mirror image} of the original tree.
- \texttt{add} adds all the numbers in the leaves of a binary tree
- \texttt{t2l} converts a tree into a (non-empty) list, preserving the left-to-right order in which the numbers appear in the tree into the left-to-right order in which they appear in the resulting list.
- \texttt{l2t} converts a (non-empty) list into a tree, preserving the left-to-right order in which the numbers appear in the list into the left-to-right order in which they appear in the resulting tree.

Even giving this left-to-right order-preservation requirement, the \texttt{l2t} \textit{is not completely determined} by such a requirement: more than one definition is possible (although one option seems the easiest to define and the most natural). That is, several trees may reasonably represent the same list preserving the left-to-right order of elements. You can give any definition you wish, provided the left-to-right order of elements is preserved.

**Notes.** (1) For defining some functions, the \texttt{if\_then\_else\_fi} operator may be useful. (2) If needed, you can also define some auxiliary functions beyond those listed above. For example, this may be helpful to define the \texttt{sort} function. (3) It may help you in testing your functions to also test that they are consistent with each other. For example, have the following built-in equality predicate expressions should all evaluate to \texttt{true} when you replace the variables \texttt{T} and \texttt{L} by any concrete trees and non-empty lists. This means that these equality predicates, viewed as \textit{equations}, are \textit{algebraic laws} that should hold between the corresponding functions mentioned in each such equality predicates.

\begin{align*}
\text{rev}(\text{t2l}(\text{T})) &= \text{t2l}(\text{trev}(\text{T})) . \\
\text{add}(\text{t2l}(\text{T})) &= \text{add}(\text{T}) . \\
\text{add}(\text{L}) &= \text{add}(\text{l2t}(\text{L})) . \\
\text{t2l}(\text{l2t}(\text{L})) &= \text{L} . 
\end{align*}

You can retrieve from the course web page this module as a “skeleton” on which to fill in your answers. Also, send a file with your solution module and your test cases to reedoei2@illinois.edu.