Programming Languages
Fall 2021

ML Assignment
Due Wednesday, November 24 at 11:55pm

Your assignment is to write in ML, using the SML/NJ system, a series of simple definitions (types and functions) for computing on lists and trees. Obviously, the code should be purely functional.

You should put your code in a file with a ".sml" extension. To load a file containing ML code into the SML/NJ system, type

```
use "filename.sml";
```

When you are finished with the assignment, submit just the file containing your definitions. Be sure to use *exactly the same function and type names* as specified in each question.

Important: Read the "Hints and Suggestions" section at the bottom of this assignment.

1. Implement an insertion function, `insert`, that takes as parameters an integer `x` and a sorted list of integers, `L`, and returns a sorted list containing all the elements of `L` along with `x` inserted in the appropriate place. For example,

   ```
   - insert 13 [8,10,12,14,16] ;
   val it = [8,10,12,13,14,16] : int list
   ```

   The function should be no more than 4 or 5 lines long.

2. Implement an insertion sort function `sort` that takes an unsorted list of integers, `L`, and returns a list containing the elements of `L` in sorted order. It must use your `insert` function above. For example,

   ```
   - sort [7,3,9,2,1,10,13,6];
   val it = [1,2,3,6,7,9,10,13] : int list
   ```

   `sort` should be 2 lines long. (z::zs)：z是个数值，zs是list

3. Implement the function `polySort`, which is a *polymorphic* version of your insertion sort function. `polySort` should sort a list of elements of any type. In order to do this, `sort` must take an additional parameter, the `<` (less-than) operator, that operates on the element types of the list. Furthermore, the definition of an insertion function, `insert` (analogous to your `insert` function, above), should be nested inside your `polySort` function (by using the `let` construct).

   For example, two uses of `polySort` would be:

   ```
   (* Using the built-in < for comparing integers. The compiler is
      smart enough to figure out which < to use *)
   - polySort (op <) [1,9, 3, 6, 7];
   val it = [1,3,6,7,9] : int list

   (* sorting a list of lists, where the less-than operator compares
      the length of two lists *)
   - polySort (fn(a,b) => length a < length b) [[1, 9, 3, 6], [1], [2,4,6], [5,5]];
   val it = [[1],[5,5],[2,4,6],[1,9,3,6]] : int list list
   ```

The `polySort` function (including the nested `polyInsert` function) should be 8 lines or less. Also, you must use the `(op <)` syntax when defining `polySort`, as follows:

```
fun polySort (op <) [] = ...
 |  polySort (op <) ... = ...
```

4. Write a function `fold` that takes a function `f`, a list `L`, and a value `b`, such that if `L` is empty, `fold` returns `b`. Otherwise, assuming `L` is of the form `[x1, x2, x3, ... xn]`, `fold` returns the result of `f(x1,f(x2,f(x3,...f(xn,b)...)))`. For example,

```
(* summing the elements of a list *)
- fold (op +) [1,2,3,4] 0;
10

(* appending together all the lists within a list of lists *)
- fold (op @) [[1,2,3,4],[5,6],[7,8,9]] [];
[1,2,3,4,5,6,7,8,9]
```

You should not use any built-in ML functions in your code for `fold`. Note also that the input type of `f` must be a tuple of two elements. An example that illustrates this is:

```
(* computing the sum of the squares of the elements of a list *)
(* notice the parameter in the lambda expression is a tuple *)
- fold (fn (x,y) => (x*x)+y) [1,2,3,4] 0;
val it = 30 : int
```

Your `fold` function should be two lines.

5. Define a polymorphic type `'a tree` using ML's datatype facility, i.e.

```
datatype 'a tree = ...
```

such that a leaf is labeled with an `'a` and an interior node has a list of children, each of type `'a tree`. That is, each interior node can have an arbitrary number of children, rather than just two (as in a binary tree). For example, your datatype declaration should allow the following tree to be constructed:

```
val myTree = node [node [node [leaf [4,2,14],leaf [9,83,32],leaf [96,123,4]],
                          node [leaf [47,71,82]],node [leaf [19,27,10],
                                                       leaf [111,77,22,66]],
                          leaf [120,42,16]],
                   leaf [83,13]]
```

The type of myTree, above, should be `int list tree`.

The datatype definition should be one line.

6. Write the `fringe` function, analogous to the one I wrote in class, that returns the fringe of an `'a tree` (above). The fringe of a tree is the list of values appearing at the leaves of a tree. For example,

```
- fringe (node [leaf 3, node [leaf 4, leaf 5], leaf 6]);
val it = [3,4,5,6] : int list
- fringe myTree;   (* see myTree above *)
val it =
  [[4,2,14],[9,83,32],[96,123,4],[47,71,82],[19,27,10],[111,77,22,66],
   [120,42,16],[83,13]] : int list list
```

Hint: Use the built-in `map` function and your `fold` function, above. Using them, `fringe` should be defined in two lines.

7. Define a polymorphic function `mapTree` that takes as parameters a function `f` and an `'a tree`, `T`, and returns a tree having the same structure as `T`, except that `f` has been applied to each element found at a leaf of `T`. For example,

```
(* This will return a tree where the integer at each leaf results from adding
   100 to the integer appearing at each leaf of the input tree *)
- mapTree (fn x => x+100) (node [leaf 1 ,leaf 2, node [leaf 3,leaf 4]]);
val it = node [leaf 101,leaf 102,node [leaf 103,leaf 104]] : int tree

(* This returns a tree where the integer at each leaf results from summing the
   list of integers at each leaf of myTree, above *)
- mapTree (fn L => fold (op +) L 0) myTree;
val it =
  node
    [node
       [node [leaf 20,leaf 124,leaf 223],node [leaf 200],
        node [leaf 56,leaf 276],leaf 178],leaf 96] : int tree
```

Assuming you are using the built-in `map` function, your `mapTree` function can be written in two or three lines.

8. Define a polymorphic `sortTree` function that, given an `'a list tree` (for some type `'a`, so that each leaf has a list of elements of type `'a`), returns a new tree that is identical to the original tree, except that the list at each leaf is sorted. It must use the polymorphic insertion sort function that you wrote (above), so therefore must also take the `<` (less-than) operator as a parameter. A use of `sortTree` is:

```
(* Here, we use the integer < to sort the list of integers at each leaf *)
- sortTree (op <) (node [leaf [5,7,3], node [leaf [9,1,4],leaf[6,10,2]]]);
val it = node [leaf [3,5,7],node [leaf [1,4,9],leaf [2,6,10]]] : int list tree

(* Here the tree is an int list list tree and the < operator compares
   the lengths of two lists *)
- sortTree (fn (L1,L2) => length L1 < length L2)
           (node [leaf [[1,2],[3],[4,5,6]],
                  leaf [[10,11,12], [13,14],[15]],
                  node [leaf [[17],[18,19,20],[21,22]]]]) ;
```

```
val it =
  node
    [leaf [[3],[1,2],[4,5,6]],leaf [[15],[13,14],[10,11,12]],
     node [leaf [[17],[21,22],[18,19,20]]]] : int list list tree
```

The `sortTree` function should be one line, assuming you use functions that you've already defined, above.

9. Define a polymorphic `mergeList` function that takes two sorted lists and returns a sorted list containing the elements of both lists. Since it is polymorphic, it also needs to take the `<` operator as a parameter. A use of `mergeList` would be:

```
- mergeList (op <) [2,4,6,8] [1,3,5,7];
val it = [1,2,3,4,5,6,7,8] : int list

(* In decreasing order *)
- mergeList (op >) [8,6,4,2] [7,5,3,1];
val it = [8,7,6,5,4,3,2,1] : int list
```

Your mergeList function should be seven lines or less.

10. Finally, define a polymorphic function `mergeTree` that, given an `'c list tree` (for some type `'c`), returns a sorted list (of type `'c list`) of all the elements found at all the leaves of the tree. It should build on code that you wrote for the previous parts of this assignment. For example, given the tree `myTree` that I defined above,

```
(* Again, myTree is an int list tree, so sortTree will return an int list *)
mergeTree (op <) myTree;
val it =
  [2,4,4,9,10,13,14,16,19,22,27,32,42,47,66,71,77,82,83,83,96,111,120,123]
  : int list
```

mergeTree should be defined in one or two lines and does NOT need to be recursive.

Hints/Suggestions

- Put the following lines at the top of your file, to tell the SML/NJ system the maximum depth of a datatype to print and the maximum length of a list to print.

```
Control.Print.printDepth := 100;
Control.Print.printLength := 100;
```

If you don't put these lines in your file, the system will only print a limited number of elements of a list, or to a limited depth in a datatype (such as a tree), after which it prints `#` to save space.

Important: The two lines above are the *only* lines in your program that where semi-colons appear. That is, you should not have semi-colons anywhere else in your program.

- The built-in `map` function in ML works just like `map` in Scheme, e.g.

```
- map (fn x => x+1) [3,4,5];
val it = [4,5,6] : int list
```