Programming Languages
Fall 2019

ML Assignment
Due Friday, November 22 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.

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

1. Implement a function `merge L1 L2`, where `L1` and `L2` are both sorted lists of integers in ascending order, that returns a sorted list containing the elements of both lists. For example,

   ```
   - merge [1,3,5,7,9] [2,4,6,8,10] ;
   val it = [1,2,3,4,5,6,7,8,9,10] : int list
   ```

   The function should be around 4-6 lines. No other function should be defined.

2. Define a function `split L`, where `L` is a list of integers, that returns a tuple of two lists, `(L1, L2)`, such that half the elements of `L` are in `L1` and half are in `L2`, in *alternating order*. For example,

   ```
   - split [1,4,2,6,8,3,9,5,4];
   val it = ([1,2,8,9,4],[4,6,3,5]) : int list * int list
   ```

   I thought of several different ways to write this, each between 4 and 9 lines of code. No external function (outside of `split`) should be defined.

3. Define a function `mergeSort L`, where `L` is a list of integers, that returns a sorted list of the elements of `L`. For example,

   ```
   - mergeSort [1,7,2,6,8,3,9,5,4];
   val it = [1,2,3,4,5,6,7,8,9] : int list
   ```

   The algorithm for `mergeSort L` is: If `L` has no more than one element, then return `L` (it is already sorted). Otherwise, split `L` into two lists (using your `split` function), recursively sort each list, and then merge the two lists back together (using your `merge`) function. No functions other than `split`, `merge`, and `mergeSort` should be called by `mergeSort`.

4. Implement the function `sort`, which is a *polymorphic* version of your merge sort function, above. `sort` 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 type of the list. Furthermore, the definition of the `split` and `merge` functions should be nested inside your `sort` function. No external function (outside of `sort`) should be defined (or called).

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

   ```
   (* Using the built-in < for comparing integers. The compiler is
      smart enough to figure out which < to use *)
   - sort (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 *)
- sort (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
```

where `length` is a built-in function. The length of your `sort` function, including the nested `split` and `merge` functions, should be roughly the same as the sum of the lengths of the functions you wrote for parts 1-3, above. Also, you must use the `(op <)` syntax when defining `sort`, as follows:

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

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

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

   such that an `'a tree` is one of the following:

   - An empty tree,
   - a leaf that is labeled with an value of type `'a`
   - an interior node that is labeled with an `'a` and has two children, each of type `'a tree`

   Your datatype declaration should allow the following tree to be constructed:

   ```
   val tree1 = node (5, node (4, leaf 3, empty),
                        node (8, node (7, leaf 6, empty),
                                 node (9, empty, leaf 10)))
   ```

   The type of `tree1`, above, should be `int tree`.

6. Write the function `labels T`, where `T` is an `'a tree`, that returns a list of the labels associated with the leaves and interior nodes of `T`. The order of the returned list should be determined by an *in-order* tree traversal, where the label of an interior node appears after all the labels of its left subtree and before all the labels of its right subtree. For example,

   ```
   - labels tree1;
   val it = [3,4,5,6,7,8,9,10] : int list
   ```

   where `tree1` is defined above.

7. Define the function `replace (op ==) x y T`, where `T` is an `'a tree`, and `x` and `y` are values of type `'a`, that returns a tree that is identical to `T` except that anywhere that a label equal to `x` appears in `T`, the label `y` appears instead. The parameter `(op ==)` is the function that should be used as the infix equality operator, so you will have to declare `==` as an infix operator. For example,

   ```
   - val tree2 = replace (op =) 4 40 tree1;  (* passing = as the == operator *)
   val tree2 =
     node
       (5,node (40,leaf 3,empty),
        node (8,node (7,leaf 6,empty),node (9,empty,leaf 10))) : int tree
   ```

```
- labels tree2;
val it = [3,40,5,6,7,8,9,10] : int list
-
- (* passing not-equals, <>, as the == operator, so any label that is not
     7 will be replaced by 0 *)
- val tree3 = replace (op <>) 7 0 tree1;
val tree3 =
  node
    (0,node (0,leaf 0,empty),
     node (0,node (7,leaf 0,empty),node (0,empty,leaf 0))) : int tree
- labels tree3;
val it = [0,0,0,0,7,0,0,0] : int list
```

8. Define the function `replaceEmpty y T`, where `T` is an `'a Tree`, which returns a new tree identical to `T`, except that each `empty` subtree in `T` has been replaced with `y`. For example,

```
  - val tree4 = replaceEmpty (node (12, leaf 11, leaf 13)) tree1 ;
val tree4 =
  node
    (5,node (4,leaf 3,node (12,leaf 11,leaf 13)),
     node
       (8,node (7,leaf 6,node (12,leaf 11,leaf 13)),
        node (9,node (12,leaf 11,leaf 13),leaf 10))) : int tree
- labels tree4;
val it = [3,4,11,12,13,5,6,7,11,12,13,8,11,12,13,9,10] : int list
```

9. Define a function `mapTree f T`, where `T` is an `'a tree`, that returns a tree resulting from applying `f` to every `node`, `leaf`, and `empty` in `T`. For example, assuming that the `increment` function is defined as

```
fun increment empty = leaf 0
 |  increment (leaf a) = leaf (a+1)
 |  increment (node (a, L, R)) = node (a+1, L, R)
```

(note that `increment` only applies to a single node and is not recursive), then

```
- val tree5 = mapTree increment tree1;
val tree5 =
  node
    (6,node (5,leaf 4,leaf 0),
     node (9,node (8,leaf 7,leaf 0),node (10,leaf 0,leaf 11))) : int tree
- labels tree5;
val it = [4,5,0,6,7,8,0,9,0,10,11] : int list
```

10. Define a polymorphic `sortTree` function that, given an `'a list tree` (for some type `'a`, so that each label is a list of elements of type `'a`), returns a new tree that is identical to the original tree, except that each label is sorted. `sortTree` must use both your `mapTree` function and your polymorphic `sort` function, above. `sortTree` should not itself be recursive. Finally, `sortTree` should use a lambda expression (see the last hint at the bottom of the assignment). For example,

```
(* Here we create a int list tree and call sortTree on it. *)
- val tree6 = node ([1,5,6,8],leaf [1,2,3,4],
                              node ([12,4,16,13], empty, leaf [0,2,5,7])) ;
val tree6 =
  node ([1,5,6,8],leaf [1,2,3,4],node ([12,4,16,13],empty,leaf [0,2,5,7]))
  : int list tree
- labels tree6;
val it = [[1,2,3,4],[1,5,6,8],[12,4,16,13],[0,2,5,7]] : int list list
- val tree7 = sortTree (op <) tree6 ;
val tree7 =
  node ([1,5,6,8],leaf [1,2,3,4],node ([4,12,13,16],empty,leaf [0,2,5,7]))
  : int list tree
- labels tree7;
val it = [[1,2,3,4],[1,5,6,8],[4,12,13,16],[0,2,5,7]] : int list list
```

Hints/Suggestions

- 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 the same function and type names as specified above.

- 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 "`#`" or "`...`" to save space. Important: The two lines above should end with the semicolons that you see. However, semicolons should not appear anywhere else in your file.

- As you know, a lambda expression defining an anonymous function in ML is written as

  `fn ... => ...`

  You can use pattern matching in a lambda expression by writing

  `fn` *pattern*$_1$ `=>` *result*$_1$ `|` *pattern*$_2$ `=>` *result*$_2$ `|` `...`

  For example,

  ```
  - map (fn 0 => 1 | 1 => 3 | n => n+10) [0,1,2,3,4]  ;
  val it = [1,3,12,13,14] : int list
  ```