CptS355 - Assignment 2 (Standard ML)
Spring 2019

Assigned: Friday, February 1, 2019

Due: Monday, February 11, 2019

Weight: Assignment 2 will count for 6% of your course grade.

Your solutions to the assignment problems are to be your own work. Refer to the course academic integrity statement in the syllabus.

This assignment provides experience in ML programming. Please compile and run your code using SML of New Jersey. You may download SML of New Jersey at http://www.smlnj.org/ .

Turning in your assignment

All code should be developed in the file called `HW2.sml`. The problem solution will consist of a sequence of function definitions in one file. Note that this is a plain text file. You can create/edit with any text editor. We recommend you to use either Visual Studio Code or Notepad++; both support syntax highlights for SML.

For each function, provide at least 2 test inputs (other than the given tests) and test your functions with those test inputs. Please see the section " Testing your functions" for more details. Include your test functions at the end of the file. Make sure that debugging code is removed before you submit your file (i.e. no print statements other than the test functions). Also, include your name as a comment at the top of the file.

To submit your assignment, turn in your file by uploading on the Assignment2 (ML) DROPBOX on Blackboard (under AssignmentSubmisions menu). You may turn in your assignment up to 3 times. Only the last one submitted will be graded.

The work you turn in is to be your own personal work. You may not copy another student's code or work together on writing code. You may not copy code from the web, or anything else that lets you avoid solving the problems for yourself. At the top of the file in a comment, please include your name and the names of the students with whom you discussed any of the problems in this homework. This is an individual assignment and the final writing in the submitted file should be *solely yours*.

Important rules

- Unless directed otherwise, you must implement your functions using recursive definitions built up from the basic built-in functions. You must program "functionally" without using ref cells (which we have not, and will not, talk about in class).
- If the problem asks for a non-recursive solution, then your function should make use of the higher order functions we covered in class (map, fold, or filter.) For those problems, your main functions can’t be recursive. If needed, you may define helper functions which are also not recursive.
- The type of your functions should match with the type specified in each problem. Otherwise you will be deducted points (around 40%). Please pay attention whether function should take the arguments in currying form vs in tuple form.

- Make sure that your function names match the function names specified in the assignment specification. **Also, make sure that your functions work with the given test cases.** However, the given test inputs don't cover all boundary cases. You should generate other test cases covering the extremes of the input domain, e.g. maximum, minimum, just inside/outside boundaries, typical values, and error values.
- Question 1(b) requires the solution to be tail recursive. Make sure that your function is tail recursive otherwise you won't earn points for this problem.
- You will call `fold, map, or filter` in several problems. Copy the definitions of `fold` and `filter` functions from the lecture slides and include them in the beginning of your file. (You can use the built-in `map`.)
- When auxiliary functions are needed, make them local functions (inside a `let` block). In this homework you will lose points if you don't define the helper functions inside a let block. The only exceptions to this are the `fold, map,` and `filter` functions.
- The assignment will be marked for good programming style (indentation and appropriate comments), as well as clean compilation and correct execution. ML comments are placed inside properly nested sets of opening comment delimiters, (*, and closing comment delimiters.*).

Problems

1. `merge2`, `merge2Tail`, and `mergeN` - 22%

(a) `merge2` - 6%

The function `merge2` takes two lists of integers, `L1` and `L2`, each already in ascending order, and returns a merged list that is also in ascending order. The resulting list should include the elements from both lists and may include duplicates.
The type of `merge2` should be `int list -> int list -> int list`.

Examples:

```
> merge2 [2,5,6,8,9] [1,3,4,5,7,8,10]
[1,2,3,4,5,5,6,7,8,8,9,10]

> merge2 [1,2] [0,10,12]
[0,1,2,10,12]

> merge2 [1,3,3,5,5] [~1,2,4]
[~1,1,2,3,3,4,5,5]

> merge2 [1,2,3] []
[1,2,3]
```

(b) `merge2Tail` - 10%

Re-write the `merge2` function from part (a) as a tail-recursive function. Name your function `merge2Tail`.
(Hint: In your bases case(s), use `revAppend` (which we defined in class) to add the reverse of the accumulated merged list to the other list.)
The type of `merge2Tail` should be `int list -> int list -> int list`.

(c) `mergeN` - 6%

Using `merge2` function defined above and the `fold` function, define `mergeN` which takes a list of lists, each already in ascending order, and returns a new list containing all of the elements in sublists in ascending order. **Provide an answer using `fold`; without using explicit recursion.**
The type of `mergeN` should be `int list list -> int list`.

Examples:

```
> mergeN [[1,2],[10,12],[2,5,6,8,9]]
[1,2,2,5,6,8,9,10,12]

> mergeN [[3,4],[~3,~2,~1],[1,2,5,8,9]]
[~3,~2,~1,1,2,3,4,5,8,9]
```

2. getInRange and countInRange - 18%

(a) getInRange – 6%

Define a function getInRange which takes two integer values, v1 and v2, and a list L, and returns the values in L which are greater than v1 and less than v2 (exclusive). **Your function shouldn't need a recursion but should use a higher order function** (map, fold, or filter). You may need to define additional helper function(s), which are also not recursive.

The type of the getInRange function should be:

```
int -> int -> int list -> int list
```

Examples:

```
> getInRange 3 10 [1,2,3,4,5,6,7,8,9,10,11]
[4,5,6,7,8,9]
> getInRange ~5 5 [~10,~5,0,5,10]
[0]
> getInRange ~1 1 [~2,2,3,4,5]
[]
```

(b) countInRange – 12%

Using getInRange function you defined in part(a) and without using explicit recursion, define a function countInRange which takes two integer values, v1 and v2, and a nested list L, and returns the total number of values in L which are greater than v1 and less than v2 (exclusive). **Your function shouldn't need a recursion but should use higher order function** (map, fold, or filter). You may need to define additional helper function(s), which are also not recursive.

The type of the countInRange function should be:

```
int -> int -> int list list -> int
```

Examples:

```
> countInRange 3 10 [[1,2,3,4],[5,6,7,8,9],[10,11]]
6
> countInRange ~5 5 [[~10,~5,~4],[0,4,5],[],[10]]
3
> countInRange 1 5 [[1,5],[1],[5],[]]
0
```

3. addLengths and addAllLengths - 18%

(a) addLengths – 10%

Define the following ML datatype which represent the US customary length units :

```
datatype lengthUnit = INCH of int | FOOT of int | YARD of int
```

Define an ML function addLengths that takes two lengthUnit values and calculates the sum of those in INCH s. (Note that 1 *foot* = 12 *inches* and 1 *yard* = 36 *inches*)

The type of the addLengths function should be:

```
lengthUnit -> lengthUnit -> lengthUnit
```

Examples:

```
> addLengths (FOOT 2) (INCH 5)
INCH 29
> addLengths (YARD 3) (INCH 3)
INCH 111
> addLengths (FOOT 3) (FOOT 5)
INCH 96
```

(b) **addAllLengths – 8%**

Now, define an ML function `addAllLengths` that takes a <u>nested</u> list of `lengthUnit` values and calculates the sum of those in `INCH` s. Your function shouldn't need a recursion but should use functions "`map`" and "`fold`". You may define additional helper functions <u>which are not recursive.</u>
The type of the `addAllLengths` function should be:

```
lengthUnit list list -> lengthUnit
```

(Hint: The `base` for `fold` needs to be a `lengthUnit` value.)

Examples:

```
> addAllLengths [[YARD 2, FOOT 1], [YARD 1, FOOT 2, INCH 10],[YARD 3]]
INCH 262
> addLengths (YARD 3) (INCH 3)
INCH 111
> addAllLengths [[FOOT 2], [FOOT 2, INCH 2],[]]
INCH 50
> addAllLengths []
INCH 0
```

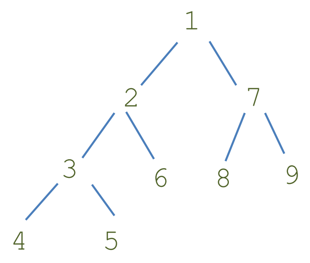
4. sumTree and createSumTree – 22%

In SML, a polymorphic binary tree type with data both at the leaves and interior nodes might be represented as follows:

```
datatype 'a Tree = LEAF of 'a | NODE of 'a * ('a Tree) * ('a Tree)
```

(a) sumTree - 7%

Write a function `sumTree` that takes a tree of type `int Tree` and calculates the sum of the values stored <u>in the leaves only</u>. The type of the `sumTree` function should be: `int Tree -> int`



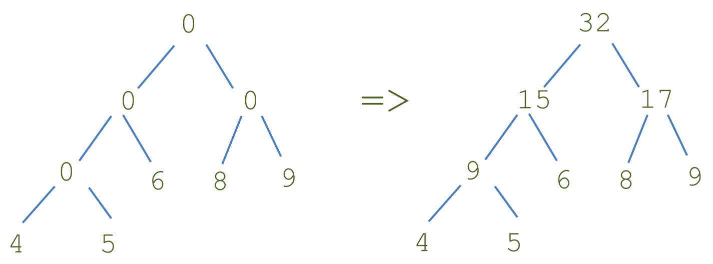
`sumTree` will return : 32

Examples:

```
> val t1 = NODE (1,
                NODE(2, NODE (3,LEAF 4, LEAF 5), LEAF 6),
                NODE(7, LEAF 8, LEAF 9))
> sumTree t1
32
> val t2 = NODE (0,
                NODE(0, LEAF 4, NODE (0,LEAF 8, LEAF 9)),
                NODE(0, NODE(0,LEAF 10, NODE (0, LEAF 12, LEAF 13)), LEAF 7))
> sumTree t2
63
> sumTree (LEAF 3)
3
```

(b) createSumTree – 15%

Write a function `createSumTree` takes an `int Tree` value and returns an `int Tree` where the interior nodes store the sum of the leaf values underneath them. See the example below.
The type of the `createSumTree` function should be `int Tree -> int Tree`



```
val t3 = NODE (0, NODE(0, NODE (0,LEAF 4, LEAF 5), LEAF 6),
                  NODE(0, LEAF 8, LEAF 9))

> createSumTree t3
```

returns

```
NODE (32, NODE (15,NODE (9,LEAF 4,LEAF 5),LEAF 6),
          NODE (17,LEAF 8,LEAF 9))
```

5. foldListTree – 20%

(16 %) A polymorphic tree type with nodes of arbitrary number of children might be represented as follows (note that the leafs store list and interior nodes store list of "`listTree`"s):

```
datatype 'a listTree = listLEAF of ('a list) | listNODE of ('a listTree list)
```

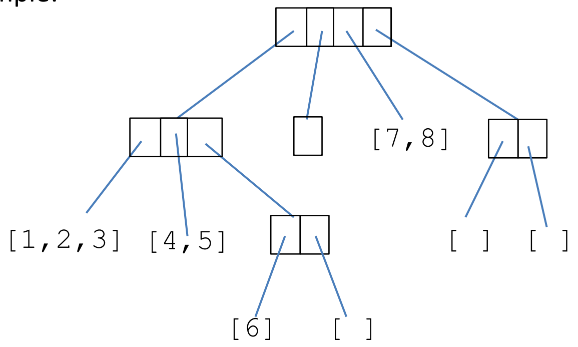
Write a function `foldListTree` that takes a function (`f`), a base value (`base`), and a listTree (`t`) and combines the values in the lists of the leaf notes in tree `t` by applying function `f`. (The leaves of the tree are scanned from left to right).
`foldListTree` is invoked as:

```
        foldListTree f base t
```

where `f` is the combining function of type `'a ->'a ->'a`.
The type of `foldListTree` should be:

```
 ('a -> 'a -> 'a) -> 'a -> 'a listTree -> 'a
```

Example:



```
val t4 = listNODE(
 [ listNODE ([ listLEAF [1,2,3],listLEAF [4,5],listNODE([listLEAF [6], listLEAF []]) ]),
   listNODE([]),
   listLEAF [7,8],
   listNODE([listLEAF [], listLEAF []]) ])

> foldListTree add 0 t4
36
```

(c) **Testing your tree functions – 4%:**
Create two tree values : one `'a Tree` and a `'a listTree` (`'a` will be substituted by the type of the values stored in the tree). The height of both threes trees should be at least 3. Test your functions `sumTree, createSumTree, foldListTree` with those trees. The trees you define should be different than those that are given. You don't need to write test functions for `sumTree, createSumTree, and foldListTree.`

SML of NJ doesn't display the full content of a tree value. To be able to display the full depth tree, you need to increase the print depth parameter. In the beginning of your file, include the following (you may also run this on the command line):
`Control.Print.printDepth := 100;`

Here is some additional test data for `foldListTree`.

```
> val L1 = listLEAF ["School","-","of","-","Electrical"]
> val L2 = listLEAF ["-","Engineering","-"]
> val L3 = listLEAF ["and","-","Computer","-"]
> val L4 = listLEAF ["Science"]
> val L5 = listLEAF ["-WSU"]
> val N1 = listNODE [L1,L2]
> val N2 = listNODE [N1,L3]
> val t5 = listNODE [N2,L4,L5]

> fun concat a b = a^b
> foldListTree concat " " t5

"School-of-Electrical-Engineering-and-Computer-Science-WSU"
```

Testing your functions

For each problem (except `sumTree`, `createSumTree`, `foldListTree`), write a test function that compares the actual output with the expected (correct) output. Below is an example test function for `addAllLengths`:

```
fun addAllLengthsTest () =
let
  val addAllLengthsT1 = ((addAllLengths [[YARD 2,FOOT 1],[YARD 1,FOOT 2,INCH 10],[YARD 3]])=(INCH 262))
  val addAllLengthsT2 = ((addLengths (YARD 3) (INCH 3)) = (INCH 111))
  val addAllLengthsT3 = ((addAllLengths [[FOOT 2], [FOOT 2, INCH 2],[]]) = (INCH 50))
 in
     print ("addAllLengths:-------------------- \n"   ^
            "   test1: " ^ Bool.toString(addAllLengthsT1) ^ "\n" ^
            "   test2: " ^ Bool.toString(addAllLengthsT2) ^ "\n" ^
            "   test4: " ^ Bool.toString(addAllLengthsT3) ^ "\n")
end
val _ = addAllLengthsTest()
```

Make sure to test your functions for at least 2 additional test inputs (in addition to the provided examples).

Hints about using files containing ML code

In order to load files into the ML interactive system you have to use the function named `use`.

The function `use` has the following syntax assuming that your code is in a file in the current directory named `HW2.sml`: You would see something like this in the output:

```
> use "HW2.sml";
[opening file "HW2.sml"]
...list of functions and types defined in your file
[closing file "HW2.sml"]
> val it = () : unit
```

The effect of use is as if you had typed the content of the file into the system, so each `val` and `fun` declaration results in a line of output giving its type.
If the file is not located in the current working directory you should specify the full or relative path-name for the file. For example in Windows for loading a file present in the users directory in the C drive you would type the following at the prompt. Note the need to use double backslashes to represent a single backslash.

```
- use "c:\\users\\example.sml";
```

Alternatively you can change your current working directory to that having your files before invoking the ML interactive system.

You can also load multiple files into the interactive system by using the following syntax

```
- use "file1"; use "file2";...; use "filen";
```

How to quit the ML environment

Control-Z followed by a newline in Windows or Control-D in Linux will quit the interactive session.