CptS355 - Assignment 2 (Standard ML)
Spring 2018

Assigned: Friday, February 9, 2018

Due: Wednesday, February 21, 2018

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. We have used both PolyML and SML of New Jersey implementations in class and you can use either for doing this assignment. You may download PolyML at http://polyml.org and SML of NewJersey at http://www.smlnj.org/ . Major Linux distributions include PolyML as an installable package (the particular version will not matter).

Turning in your assignment

All code should be developed in the file called HW2.sml. Implement the functions for the given problems and include them in HW2.sml file. Note that this is a plain text file. You can create/edit with any text editor.

For each function, provide at least 3 test inputs (other than the given tests) and test your functions with those test inputs. Please see the section below "Test 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 4 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.

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 recursive. However, some functions may require the helper functions to be non-recursive as well.
- The type of your functions should match with the type specified in each problem. Each problem specifies the number of arguments the functions should take and the type of each argument.

You function definition should comply with that specification. Otherwise you will be deducted points (around 40%).

- 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.**
- Some questions require the solution to be tail recursive. Make sure that your function is tail recursive for those problems.
- You will call `fold` and `map` in several functions. Copy the definitions of `fold` and `map` functions from the lecture slides and include them in the beginning of your file.
- 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 `map` and `fold` 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. countInList, zipTail, histogram

(a) countInList [tail recursive]- 5%

`countInList` function takes a value and a list and returns the number of occurrences of that value in the input list. The type of your function should be `''a list -> ''a -> int`.
For this problem, your implementation **is required to be tail recursive.**

Examples:

```
> countInList ["3","5","5","-","4","5","1"] "5"
3
> countInList [] "5"
0
> countInList [true, false, false, false, true, true, true]  true
4
> countInList [[],[1,2],[3,2],[5,6,7],[8],[]]  []
2
```

(b) zipTail [tail recursive]- 10%

`zipTail` function takes two lists, pairs up the corresponding elements from two lists, and returns a merged list of tuples. The input lists don't necessarily need to be of equal length. The additional elements in the longer list will be ignored.
The type of the `zipTail` function should be `'a list -> 'b list -> ('a * 'b) list`.
For this problem, your implementation **is required to be tail recursive,** so you will need to define an auxiliary function that has an extra parameter in which the result is accumulated. It turns out that in using the accumulating parameter technique, the result is produced in reverse order. So you also need to define the function `reverse` that reverses a list. `reverse` should also be implemented as a tail-recursive function.

Examples:

```
> zipTail [1,2,3,4,5] ["one","two"]
[(1,"one"),(2,"two")]
> zipTail [1] [1,2,3,4]
[(1,1)]
> zipTail [1,2,3,4,5] []      (*may give a warning*)
[]
> zipTail [] [1,2,3,4,5]      (*may give a warning*)
[]
```

(c) histogram - 10%

`histogram` function takes a list as input and returns a list of tuples where the first elements in the tuples are the unique elements from the input list and the second elements are the number of occurrences of those elements in the tuple. Your function shouldn't need a recursion but should use "`map`" function. A possible solution uses "`map`", "`countInList`","`zipTail`", "`inList`", and "`removeDuplicates`".

"countInList" and "zipTail" are the functions you defined in parts (a) and (b). "inList", and "removeDuplicates" are the functions you defined in assignment-1. Sample solutions of those are available below.
The type of the histogram function should be: ''a list -> (''a * int) list.

Examples: *(Note: Order of the tuples in your output can be different than the order in the given sample output)*

```
> histogram [1,3,2,2,3,0,3]
[(1,1),(2,2),(0,1),(3,3)]
> histogram [[1,2],[3],[],[3],[1,2]]
[([],1),([3],2),([1,2],2)]
> histogram []                    (*may give a warning*)
[]
> histogram [true, false, false, false, true, true, true]
[(false,3),(true,4)]
```

```
fun inList (n,[]) = false
 |  inList(n,x::rest) = if n=x then true else inList(n,rest)

fun removeDuplicates [] = []
 |  removeDuplicates (x::rest) = if inList(x,rest) then (removeDuplicates rest)
                                 else x::(removeDuplicates rest)
```

2. deepSum and deepSumOption

(a) deepSum - 5%

Function deepSum is given a list of int lists and it returns the sum of all numbers in all sublists of the input list. Your function shouldn't need a recursion but should use functions "map" and "fold". You may define additional helper functions which are not recursive.
The type of the deepSum function should be int list list -> int.
Examples:

```
> deepSum [[1,2,3],[4,5],[6,7,8,9],[]]
45
> deepSum [[10,10],[10,10,10],[10]]
60
> deepSum [[]]
0
> deepSum []
0
```

(b) deepSumOption - 15%

Function deepSumOption is given a list of int option lists and it returns the sum of all int option values in all sublists of the input list. Your function shouldn't need a recursion but should use functions "map" and "fold". You may define additional helper functions which are not recursive. The type of the deepSumOption function should be:
int option list list -> int option.

(Note: To implement deepSumOption, change your deepSum function and your helper function in order to handle int option values instead of int values. Assume the integer value for NONE is 0.)

Examples:

```
> deepSumOption [[SOME(1),SOME(2),SOME(3)],[SOME(4),SOME(5)],[SOME(6),NONE],[],[NONE]]
SOME 21
> deepSumOption [[SOME(10),NONE],[SOME(10), SOME(10), SOME(10),NONE,NONE]]
SOME 40
> deepSumOption [[NONE]]
NONE
> deepSumOption []
NONE
```

3. unzip – 10%

For this problem, define the inverse of `zip` **without using recursion**, namely `unzip`, that takes a list of tuples as input and produces a list including two lists as output. (Hint: Call the `map` function twice to get the first and second values from pairs). The type of the `unzip` function should be:

```
('a * 'a) list -> 'a list list
```

What should `(unzip [])` be?

Examples:

```
> unzip [(1,2),(3,4),(5,6)]
[[1,3,5],[2,4,6]]
>  unzip [("1","a"),("5","b"),("8","c")]
[["1","5","8"],["a","b","c"]]
```

4. eitherTree and eitherSearch - 20%

a) Define the following ML datatype

```
datatype either = ImAString of string | ImAnInt of int
```

b) Define ML datatype named `eitherTree` for binary trees containing values of type `either` where data may be held at both interior and leaf nodes of the tree. (for leaf nodes use `eLEAF of either` and for interior nodes use `eINTERIOR of (either*eitherTree*eitherTree) )`.

c) Define an ML function `eitherSearch` that takes an integer value and an `eitherTree` and returns `true` if the `integer` value is in the tree and returns false otherwise. The trick to getting this to type check is to realize that `ImAnInt` of `int` values and `int` values do not have the same type. But you can transform either into the other. The type of the `eitherSearch` function should be:

```
eitherTree -> int -> bool
```

and

d) Define an ML function `eitherTest` that takes an argument and:

- constructs an eitherTree with at least 5 `int`-containing leaves, at least 5 string-containing leaves, and at least 4 levels;
- searches the tree using your `eitherSearch` function for an `int` that is not present in the tree.

(Note: You don't need to provide additional test function for `eitherSearch`. `eitherTest` will be considered your test function.)

5. `findMin, findMax, minMaxTree`

(a) `findMin, findMax` - 10%

In SML, a polymorphic binary tree type with data only at the leaves might be represented as follows:

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

And a polymorphic binary tree type with data on both leaves and nodes might be represented as follows (note that each `myNODE` stores 2 values) :

```
datatype 'a myTree = myLEAF of 'a | myNODE of 'a*'a*('a myTree)*('a myTree)
```

Write the functions `findMin` and `findMax`, which take a tree of type `'a Tree` and return the minimum and maximum values stored in the leaves of the tree, respectively. The type of both functions should be: `int Tree -> int`.

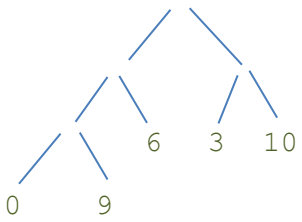
Examples:

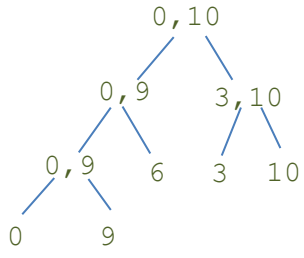
```
> findMin (NODE(NODE(LEAF(5),NODE(LEAF(6),LEAF(8))),LEAF(4)))
4
> findMin (NODE(NODE(NODE(LEAF(0),LEAF(11)),LEAF(6)),NODE(LEAF(3),LEAF(10))))
0
> findMin (LEAF(5))
5

> findMax (NODE(NODE(LEAF(5),NODE(LEAF(6),LEAF(8))),LEAF(4)))
8
> findMax (NODE(NODE(NODE(LEAF(0),LEAF(11)),LEAF(6)),NODE(LEAF(3),LEAF(10))))
11
> findMax (LEAF(5))
5
```

(b) `minmaxTree` - 15%

Define a function `minmaxTree` that takes a tree of type `int Tree` as argument and returns another tree of type `int myTree`. In the returned tree, each node includes the min and max values that appear in the leaves of the subtree rooted at that node. For example, `minmaxTree` will take the `intTree` on the left as input, and it will generate the `int myTree`, given on the right.





We can create the `int Tree` on the left as follows:

```
(NODE(NODE(NODE(LEAF(0),LEAF(9)),LEAF(6)),NODE(LEAF(3),LEAF(10))))
```

And `minmaxTree` will return the `int myTree` on the right which is equivalent to the following:

```
(myNODE(0,10,myNODE(0,9,myNODE(0,9,myLEAF(0),myLEAF(9)),myLEAF(6)),myNODE(3,10,myLEAF(
3),myLEAF(10))))
```

The type of the `minmaxTree` function should be `int Tree -> int myTree`.

(c) Create two trees of type `int Tree`. The height of both trees should be at least 4. Test your function `minmaxTree` with those trees. The trees you provide should be different than those that are given (see above and below examples).

Testing `minmaxTree`:
Both SML of NJ and Poly ML doesn't display the full content of a tree value. To make sure your output is correct, you may create a function to print the content of the tree and make sure your `minmaxTree` function returns the correct tree.

To test your `minmaxTree` function simply call your function for the two trees you created in part (c). You don't need to write additional test functions.

Here is some additional test data:

```
val L1 = LEAF(1)
val L2 = LEAF(2)
val L3 = LEAF(3)
val N1 = NODE(L1,L2)
val N2 = NODE(N1,L1)
val N3 = NODE(N1,N2)
val t1 = NODE(N2,N3)

minmaxTree t1
```

Extra Credit: Priority Queue - Abstact Data Type (5%)

Priority queue is a widely-used data structure. It is an ordinary queue extended with an integer priority. When data values are added to a queue, the priority controls where the value is added. A value added with priority p is placed behind all entries with a priority = p and in front of all entries with a priority > p. Note that if all entries in a priority queue are given the same priority, then a priority queue acts like an ordinary queue in that new entries are placed behind current entries. You are to write an ML abstract data type (an `abstype`) that implements a polymorphic priority queue, defined as 'a `PriorityQ`. You may implement your priority queue using any reasonable ML data structure (a list of tuples might be a reasonable choice). The following values, functions, and exceptions should be implemented:

- `exception emptyQueue`
  This exception is raised when front or remove is applied to an empty queue.
- `nullQueue`
  This value represent the null priority queue, which contains no entries.
- `enter(pri,v,pQueue)`
  This function adds an entry with value `v` and priority `pri` to `pQueue`. The updated priority queue is returned. As noted above, the entry is placed behind all entries with a priority `= pri` and in front of all entries with a priority > `pri`.
- `front(pQueue)`
  This function returns the front value in `pQueue`, which is the value with the lowest priority. If

more than one entry has the lowest priority, the oldest entry is chosen. If `pQueue` is empty, the `emptyQueue` exception is raised.

- `remove(pQueue)`
  This function removes the front value from `pQueue`, which is the value with the lowest priority. If more than one entry has the lowest priority, the oldest entry is removed. The updated priority queue is returned. If `pQueue` is empty, the `emptyQueue` exception is raised.
- `contents(pQueue)`
  This function returns the contents of `pQueue` in list form. Each member of the list is itself a list comprising all queue members sharing the same priority. Sublists are ordered by priority, with lowest priority first. Within a sublist, queue members are ordered by order of entry, with oldest first. The front of `pQueue` is the leftmost element of the first sublist, and the rear of `pQueue` is the rightmost member of the last sublist.

Testing your functions

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

```
fun zipTailTest () =
 let
     val zipT1 = ((zipTail [1,2,3,4,5] ["one","two"]) = [(1,"one"),(2,"two")])
     val zipT2 = ((zipTail [1] [1,2,3,4]) = [(1,1)])
     val zipT3 = ((zipTail [1,2,3,4,5] []) = [])
     val zipT4 = ((zipTail [] [1,2,3,4,5]) = [])
 in
     print ("zipTail:-------------------- \n test1: " ^ Bool.toString(zipT1) ^
            "   test2: " ^ Bool.toString(zipT2) ^
            "   test3: " ^ Bool.toString(zipT3) ^
            "   test4: " ^ Bool.toString(zipT4) ^ "\n")

end
val _ = zipTailTest()
```

Make sure to test your functions for at least 3 additional test cases (in addition to the provided examples).

Note that for `eitherSearch`, `eitherTest` will be your test function. For `minmaxTree`, you don't need to provide a test function (you only need to create two `int Tree`'s and call your function with those).

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 "HW4.sml"]
...list of functions and types defined in your file
[closing file "HW4.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.

ML resources:

- PolyML
- Standard ML of New Jersey
- Moscow ML
- Prof Robert Harper's CMU SML course notes