CptS355 - Assignment 4 (Standard ML)
Spring 2017

Assigned: Tuesday March 21, 2017

Due: Monday April 3, 2017 (5% extra credit if you submit by March 31st)

Weight: Assignment 4 will count for 10% 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. You have previously written some of these functions in Scheme. Now you get to see how they appear in a statically type-checked language. 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 `HW4.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. To submit your assignment, turn in your file by uploading on the Assignment4 (ML) DROPBOX on Blackboard (under AssignmentSubmisions menu). Include code (functions) that you use to test the required functions, but make sure that debugging code is removed from the required functions themselves (i.e. no print statements other than those that print the final output). Please see the section below "Test Functions" for more details. Be sure to include your name as a comment at the top of the file. You may turn in your assignment up to 5 times. Only the last one submitted will be graded. Please let the instructor know if you need to resubmit it a 6th time.

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.

Grading

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,*).

General rules

Unless directed otherwise, you must implement your functions using recursive definitions built up from the basic built-in functions (i.e., when I ask you to implement filter you may not use the built-in filter function, nor can you use the built-in sort functions for the sorting assignment, etc.). You must program "functionally" without using ref cells (which we have not, and will not, talk about in class).

When auxiliary functions are needed, make them local functions (inside a `let` block).

Problems

1. `exists` – 10%

This function should return true if the first argument is a member of the second argument and have type `(''a * ''a list) -> bool`. Explain in a comment why the type `is (''a * ''a list) -> bool` and not `('a * 'a list) -> bool`
Note: this function is not required to be tail-recursive.

Examples:

```
> exists (1,[]);
false
> exists (1,[1,2,3]);
true
> exists ([1],[[1]]);
true
> exists ([1],[[3],[5]]);
false
> exists ("c",["b","c","z"]);
true
```

2. `listUnion` - 10%

This function should return the union of two lists. Each value should appear in the output list only once but the order does not matter. In the `listIntersect` function below the two lists were supplied as separate arguments. Make `listUnion` a function of a single argument with that argument being a tuple: it should have type `(''a list * ''a list) -> ''a list`.
Note: this function is not required to be tail-recursive.

Examples:

```
> listUnion ([1], [1]);
[1]
> listUnion ([1,1,3], [1,2]);
[2,1,3]
> listUnion ([[2,3],[1,2]], [[1],[2,3]]);
[[2,3],[1],[1,2]]
```

3. `listIntersect` - 10%

This function should return the intersection of two lists, and have type `''a list -> ''a list -> ''a list`.
The duplicates should be eliminated in the result. (Hint: You can make use of `exists`.)
Note: this function is not required to be tail-recursive.

Examples:

```
> listIntersect [1] [1];
[1]
> listIntersect [1,2,3] [1,1,2];
[1,2]
> listIntersect [[2,3],[1,2],[2,3]] [[1],[2,3]];
[[2,3]]
```

4. **`pairNleft and pairNright - 15%`**

These functions take two arguments. The first is an integer and the second is a list. The idea is to produce a result in which the elements of the original list have been collected into sublists each containing N elements (where N is the integer argument). Thus the type of each of these is `int -> 'a list -> 'a list list`. The difference between the two functions is how they handle the left-over elements of the list when the integer doesn't divide the length of the list evenly. `pairNleft` treats the initial elements of the list as the extras, thus the result starts with a list of between 1 and N elements and all the remaining elements of the result list contain N elements. `pairNright` does the opposite: the final group contains between 1 and N elements and all the rest contain N elements. Note: these functions are not required to be tail-recursive.

Examples:

```
> pairNleft 2 [1, 2, 3, 4, 5]
[[1], [2, 3], [4, 5]]
> pairNright 2 [1, 2, 3, 4, 5]
[[1, 2], [3, 4], [5]]
> pairNleft 3 [1, 2, 3, 4, 5]
[[1, 2], [3, 4, 5]]
> pairNright 3 [1, 2, 3, 4, 5]
[[1, 2, 3], [4, 5]]]
```

5. **`filter (and reverse) - 10%`**

`filter` takes a one-argument function (called a predicate - which returns true or false) and a list, and it returns a list of all elements in the input list that satisfy that predicate. The elements must appear in the result in the same order that they appear in the original list.

Examples:

```
> filter (fn (x) => (x = 1)) [1,2,3];
[1]
> filter (fn (x) => (x <= 3))[1,2,3,4];
[1,2,3]
```

For this problem (only) your implementation is required to **be tail recursive**, so you will need to define an auxiliary function that takes a parameter in which the result is accumulated. (It is the auxiliary function that will be tail recursive. The function filter will simply call the auxiliary function with `[]` as the initial result.) 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. In class when talking about Scheme, we defined `revAppend` and then `reverse`. Translate those definitions to ML.

We will talk about tail recursion and accumulating parameters in class. We will cover examples of how to write auxiliary functions as nested functions.

6. **Merge Sort - 15%**

a) Assume we have a list L of integers. Define a function `unitList L` that places each integer in L into its own sublist (of size one). That is, if L has n integers in it, `unitList` produces a list of n sublists, each containing a single integer from L. For example,

```
> unitList []
[]
> unitList [1,2,3,4]
[[1],[2],[3],[4]]
```

b) Each of the sublists produced by unit-lists is trivially sorted. If we merge the first two sublists together into sorted order, we have a sorted sublist of size 2. If we then merge the third and fourth sublists, then the fifth and sixth sublists, etc., we end up with n/2 sorted sublists of size 2, rather than n sublists of size 1 (the final sublist may be of size 1 if it has no partner to pair with). If we iterate this merging process, we next get n/4 sorted sublists of size 4, then n/8 sorted sublists of size 8, etc. Finally, we produce a single sorted sublist of size n. This sorting logic is the basis of a merge sort. Write an ML function `mergeSort(L)` that first divides L into unit sublists using `unitList`, and then repeatedly merges adjacent sublists until a single sorted list is produced. You may create and use any additional functions you find useful or necessary.

```
> mergeSort [5,3,6,3,1,7,2,4,1];
[1,1,2,3,3,4,5,6,7]
```

c) Once a list is sorted, it easy to test for duplicates—just compare adjacent values. But testing for duplicates after the sort is finished is somewhat inefficient. If a duplicate appears in L it can readily be detected when values from sublists are merged. Create a function `mergeSort2(L)` that removes duplicates while merging.

```
> mergeSort2 [5,3,6,3,1,7,2,4,1];
[1,2,3,4,5,6,7]
```

7. **eitherTree and eitherSearch - 15%**

a) Define an ML datatype

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

b) Define an 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 interior nodes use (`either * eitherTree * eitherTree`)).

c) Define an ML function `eitherSearch` having type `eitherTree -> int -> bool` that returns `true` if the `int` is in the tree and 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.

d) Define an ML function of no arguments, `eitherTest` that:

- 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 present in the tree;
- and, searches the tree using your `eitherSearch` function for a value that is not present in the tree.

8. `treeToString` - 15%

A polymorphic tree type, with data only at the leaves, in SML might be represented using

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

Write a function `treeToString: ('a -> string) -> 'a Tree -> string` that returns a parenthesized string representing an arbitrary `Tree`. `treeToString` is invoked as:

```
treeToString f t
```

where `f` is a function that converts data of type `'a` to a `string` and `t` is an `'a Tree`. The parenthesization rules implemented by `treeToString` are as follows:

- For a LEAF node, the returned value is just : `(f the-data-in-the-leaf)`.
- For a LIST node, concatenate the strings produced by `treeToString` on the elements of the list and surround the resulting string with parentheses.

Notes:

- For this function, you may use built-in functions `map` and `String.concat` in addition to the generally allowable functions listed above.
- I suggest that you start by solving a simpler non-polymorphic problem using
  `datatype Tree = LEAF of string | NODE of Tree list`
  In the simpler problem since the leaves are already strings, you won't need the function argument (that converts `'a` to `string`). Once you make the simpler version work, make the data type polymorphic and add the function parameter.
  - Hint: The whole function can be expressed as two lines of code.

Testing `tree2String`:

Here is some test data:

```
val L1a = LEAF "a"
val L1b = LEAF "b"
val L1c = LEAF "c"
val L2a = NODE [L1a, L1b, L1c]
val L2b = NODE [L1b, L1c, L1a]
val L3 = NODE [L2a, L2b, L1a, L1b]
val L4 = NODE [L1c, L1b, L3]
val L5 = NODE [L4]

val iL1a = LEAF 1
val iL1b = LEAF 2
val iL1c = LEAF 3
val iL2a = NODE [iL1a, iL1b, iL1c]
val iL2b = NODE [iL1b, iL1c, iL1a]
```

```
val iL3 = NODE [iL2a, iL2b, iL1a, iL1b]
val iL4 = NODE [iL1c, iL1b, iL3]
val iL5 = NODE [iL4]
```

Examples:
`treeToString String.toString L5` should produce `"((cb((abc)(bca)ab)))"`

`treeToString Int.toString iL5` should produce `"((32((123)(231)12)))"`.

Additional Notes:

- Note that interactive SML systems typically do not print all of the contents of deeply nested data structures. So after evaluating the declaration for iL5 the response may be something like
  `val iL5 = NODE [NODE [LEAF #,LEAF #,NODE #]] : int Tree`
  depending on what SML system you are using (in this case SMLofNJ).

- Additional information about string manipulation:
  - the ^ infix operator concatenates two strings, thus:
    ```
    > "abc" ^ "def";
    "abcdef"
    ```
  - String.concat concatenates all of the strings in a list of strings (look at it's type!). Thus:
    ```
    > String.concat ["abc", "def", "ghi"];
    "abcdefghi"
    ```

Extra Credit: Priority Queue - Abstact Data Type (10%)

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 are example test functions for `pairNleft` and `pairNright`:

```
fun myTest_pairNleft (n,L,R) = if (pairNleft n L) = R then true else
false
fun myTest_pairNright (n,L,R) = if (pairNright n L) = R then true else
false

val test1_result = myTest_pairNleft(2,[1,2,3,4,5],[[1],[2,3],[4,5]])
val test2_result = myTest_pairNright(2,[1,2,3,4,5],[[1,2],[3,4],[5]])
```

Make sure to test your functions for at least 3 test cases.

Note that for problem-7, `eitherTest` will be your test function.

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 `HW4.sml`: You would see something like this in the output:

```
> use "HW4.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