CptS355 - Assignment 1 (Standard ML)
Fall 2018

Assigned: Tuesday, September 4, 2018

Due: Wednesday, September 12, 2018

Weight: Assignment 1 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 SML of New Jersey and PolyML 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 `HW1.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.

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 Assignment1 (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 are not allowed to import an external library and use functions from there.) You must program "functionally" without using ref cells (which we have not, and will not, talk about in class).
- 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.** 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.

- 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 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. **exists and countInList – 15%**

a) **[6pts]** Write a function **`exists`** which takes a tuple as input where the first element of the tuple is a value and the second is a list. If the value is a member of the list, the function should return `true`. Otherwise it should return `false`.
The function should have type `(''a * ''a list) -> bool`.

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
```

b) **[2pts]** Explain in a comment why the type is
`(''a * ''a list) -> bool` and not
`('a * 'a list) -> bool`

c) **[7pts]** Write a function **`countInList`** which 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.`

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
```

2. **listDiff - 15%**

Write a function `listDiff` that takes a single argument (a 2-tuple) where both elements of the tuple are lists. The function should return the difference of the first list with respect to the second. You should treat the lists as bags (i.e., they can have duplicates). If an element appears in both lists and if the number of duplicate copies of the element is bigger in the first list, then this element should appear in the result as many times as the difference of the number of occurrences in the input lists.

Your function should have type `(''a list * ''a list) -> ''a list.` The duplicates should not be eliminated in the result. The elements in the resulting list may have arbitrary order.
(Hint: You can make use of `countInList` function.)

Examples:

```
> listDiff (["a","b","c"],["b"])
["a", "c"]
> listDiff ([1,2,3],[1,1,2])
[3]
> listDiff ([1,2,2,3,3,3],[1,1,2,3])
[2,3,3]
> listDiff ([[2,3],[1,2],[2,3]],[[1],[2,3]])
[[2,3],[1,2]]
> listDiff ([1,2,3],[])
[1,2,3]
```

3. **firstN – 13%**

Write a function `firstN` that takes a list and an `int n` and returns the first `n` elements in the list. The type of `firstN` should be `string 'a list -> int -> 'a list.`
If the input list is empty or if the length of the list is less than `n`, then the function should return the complete list. (You may assume that `n` is greater than 0.)

Examples:

```
> firstN ["a", "b", "c", "x", "y"] 3
["a", "b", "c"]
> firstN [1,2,3,4,5,6,7] 4
[1,2,3,4]
> firstN [1,2,3,4,5,6,7] 10
[1,2,3,4,5,6,7]
> firstN [[1,2,3],[4,5],[6],[],[7,8],[]] 4
[[1,2,3],[4,5],[6],[]]
> firstN [] 5    /*may give a warning*/
[]
```

4. **busFinder – 17%**

Pullman Transit offers many bus routes in Pullman. Assume that they maintain the bus stops for their routes as a list of tuples. The first element of each tuple is the bus route and the second element is the list of stops for that route (see below for an example).

```
val buses = [
("Lentil",["Chinook", "Orchard", "Valley", "Emerald","Providence",
"Stadium", "Main", "Arbor", "Sunnyside", "Fountain", "Crestview",
"Wheatland", "Walmart", "Bishop", "Derby", "Dilke"]),
("Wheat",["Chinook", "Orchard", "Valley", "Maple","Aspen",
"TerreView", "Clay", "Dismores", "Martin", "Bishop", "Walmart",
"PorchLight", "Campus"]),
("Silver",["TransferStation", "PorchLight", "Stadium",
"Bishop","Walmart", "Shopco", "RockeyWay"]),
("Blue",["TransferStation", "State", "Larry", "TerreView","Grand",
"TacoBell", "Chinook", "Library"]),
("Gray",["TransferStation", "Wawawai", "Main",
"Sunnyside","Crestview", "CityHall", "Stadium", "Colorado"])
]
```

a) Assume that you are creating an application for Pullman Transit. You would like to write an ML function `busFinder` that takes the list of bus routes and a stop name, and returns the list of the bus routes which stop at the given bus stop.
The type of `busFinder` should be `''a -> ('b * ''a list) list -> 'b list.`
(Hint: You can make use of `exists` function you defined earlier.)

Examples:

```
> busFinder "Walmart" buses
["Lentil","Wheat","Silver"]

> busFinder "Shopco" buses
["Silver"]

> busFinder "Main" buses
["Lentil","Gray"]

> busFinder "Beasley" buses
[]
```

b) **[2pts]** Explain in a comment why the type is
`''a -> ('b * ''a list) list -> 'b list` and not
`''a -> ('a * ''a list) list -> 'a list`

5. `parallelResistors` – 15%

Write a function `parallelResistors` that calculates the total resistance of a number of resistors connected in parallel. The formula for computing this is:

$$x = \frac{1}{\frac{1}{r1} + \frac{1}{r2} + \cdots + \frac{1}{rn}}$$

where `r1, . . . rn` are the resistances of each resistor.
For example, `parallelResistors [10.0, 10.0]`, representing two 10.0 Ohm resistors in parallel, should return `5.0`.
Don't worry about the edge cases of zero Ohm resistors, or no resistors (i.e. an empty list as an argument to `parallelResistors`). However, after you've written your function, try it on these cases and see what happens. **Explain why you get the results that you do in a comment [2 pts].**
The type of `parallelResistors` should be `real list -> real.`

Examples:

```
> parallelResistors [10.0, 10.0, 10.0, 10.0]
2.5
> parallelResistors [8.0, 16.0, 4.0, 16.0]
2.0
> parallelResistors [5.0, 10.0, 2.0]
1.25
```

6. `pairNleft and pairNright` - 25%

These functions takes a 2-tuple (e.g. `(N,L)`) where the first element is an integer (N) and the second is a list (`L`). The idea is to produce a result in which the elements of the original list have been collected

into lists each containing n elements (where `N` is the integer argument). The leftover elements (if there is any) are included as a tuple with less than `N` elements.
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]]]
> pairNright (3,[])     (*this may give a warning since the type can't be inferred*)
[[]]
```

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 `groupNleft` and `groupNright`:

```
fun busFinderTest () =
let
  val buses =
      [("Lentil",["Chinook", "Orchard", "Valley", "Emerald","Providence", "Stadium",
      "Main", "Arbor", "Sunnyside", "Fountain", "Crestview", "Wheatland", "Walmart",
      "Bishop", "Derby", "Dilke"]),
      ("Wheat",["Chinook", "Orchard", "Valley", "Maple","Aspen", "TerreView", "Clay",
      "Dismores", "Martin", "Bishop", "Walmart", "PorchLight", "Campus"]),
      ("Silver",["TransferStation", "PorchLight", "Stadium", "Bishop","Walmart",
      "Shopco", "RockeyWay"]),
      ("Blue",["TransferStation", "State", "Larry", "TerreView","Grand", "TacoBell",
      "Chinook", "Library"]),
      ("Gray",["TransferStation", "Wawawai", "Main", "Sunnyside","Crestview",
      "CityHall", "Stadium", "Colorado"])]

  val busFinderT1 = ((busFinder "Walmart" buses) = ["Lentil","Wheat","Silver"])
  val busFinderT2 = ((busFinder "Shopco" buses) = ["Silver"])
  val busFinderT3 = ((busFinder "Main" buses) = ["Lentil","Gray"])

 in
    print ("busFinder:-------------------- \n"   ^
           "   test1: " ^ Bool.toString(busFinderT1) ^ "\n" ^
           "   test2: " ^ Bool.toString(busFinderT2) ^ "\n" ^
           "   test4: " ^ Bool.toString(busFinderT3) ^ "\n")
end
val _ = busFinderTest ()
```

Make sure to test your functions for at least 2 test cases (in addition to the provided examples). Test functions should be included at the end of your HW1.sml file.

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 "HW1.sml";
[opening file "HW1.sml"]
...list of functions and types defined in your file
[closing file "HW1.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