CptS355 - Assignment 1 (Standard ML)
Spring 2018

Assigned: Wednesday, January 17, 2018

Due: Wednesday, January 31, 2018

Weight: Assignment 1 will count for 5% 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 `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. To submit your assignment, turn in your file by uploading on the Assignment1 (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 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.

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. 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. `inList` – 10%

a) Write a function `inList` 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`.

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

Examples:

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

2. `removeDuplicates` - 10%

Write a function `removeDuplicates` that takes a list as input and removes the duplicate values from the list. Each value should appear in the output list only once but the order does not matter. The function should have type `''a list -> ''a list`.

Examples:

```
> removeDuplicates [1, 5, 1, 3, 4, 3, 5];
[1,4,3,5]
> removeDuplicates ["a", "e", "c", "a", "a", "b", "c", "d"];
["e","a","b","c","d"]
> removeDuplicates [];   (*this may give a warning*)
[]
```

3. `listIntersect` - 10%

Write a function `listIntersect` that takes a single argument (a 2-tuple) where both elements of the tuple are lists. The function should return the intersections of these lists and should have type `(''a list * ''a list) -> ''a list.` The duplicates should be eliminated in the result. (Hint: You can make use of `inList` and `removeDuplicates`)

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]]
```

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

4. `range` – 15%

Write a function `range` that takes a starting index `min`, a step value `step`, and a max index `max` and returns a list of integers `[min, min+step, min+2*step, … ,min+k*step]` where k is the largest integer such that `(min+k*step)<max` and `(min+(k+1)*step)≥max`. If `min≥max` and `step≥0` OR `min≤max` and `step<0`, then the function should return an empty list.
This function should have type `int -> int -> int -> int list`

Examples:

```
> range 0 5 30;
[0,5,10,15,20,25]
> range 10 1 10;
[]
> range 5 ~1 0
[5,4,3,2,1]
> range 1 ~1 10;
[]
```

5. `numbersToSum` – 15%

Write a function `numbersToSum` that takes an `int` value (called `sum` - which you can assume is positive), and an `int list` (called `L` - which you can assume contains positive numbers) and returns an `int list`. The returned list should include the first `n` elements from the input list `L` such that the first `n` elements of the list add to less than `sum`, but the first `(n + 1)` elements of the list add to `sum` or more. Assume the entire list sums to more than the passed in `sum` value.
The type of `numbersToSum` should be `int -> int list -> int list.`

Examples:

```
> numbersToSum 100 [10, 20, 30, 40]
[10, 20, 30]
> numbersToSum 30 [5, 4, 6, 10, 4, 2, 1, 5]
[5, 4, 6, 10, 4]
> numbersToSum 1 [2];
[]
> numbersToSum 1 [];
[]
```

6. `replace` – 15%

Write a function `replace` that takes an index **n**, a value **v**, and a list **L** and returns a (new) list which is the same as **L**, except that **n**th element is **v**. Assume 0 based indexing. Also assume that the index, **n**, is at least `0` and smaller than the length of the list **L**.
The type of `replace` should be `int -> 'a -> 'a list -> 'a list.`
Examples:

```
> replace 3 40 [1, 2, 3, 4, 5, 6]
[1,2,3,40,5,6]
> replace 0 "X" ["a", "b", "c", "d"];
["X","b","c","d"]
> replace 4 false [true, false, true, true, true]
```

```
[true,false,true,true,false]
```

7. groupNleft and groupNright - 25%

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. `groupNleft` 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. `groupNright` 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:

```
> groupNleft 2 [1, 2, 3, 4, 5];
[[1], [2, 3], [4, 5]]
> groupNright 2 [1, 2, 3, 4, 5];
[[1, 2], [3, 4], [5]]
> groupNleft 3 [1, 2, 3, 4, 5];
[[1, 2], [3, 4, 5]]
> groupNright 3 [1, 2, 3, 4, 5];
[[1, 2, 3], [4, 5]]]
```

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 myTest_groupNleft (n,L,output) =
               if (groupNleft n L) = output then true else false
fun myTest_groupNright (n,L,output) =
               if (groupNright n L) = output then true else false

val groupNleft_test1 = myTest_groupNleft(2,[1,2,3,4,5],[[1],[2,3],[4,5]])
val groupNright_test1 = myTest_groupNright(2,[1,2,3,4,5],[[1,2],[3,4],[5]])
```

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

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