CptS355 - Assignment 1 (Standard ML)
Spring 2019

Assigned: Wednesday January 16, 2019

Due: Monday January 28, 2019 11:59pm

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. 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 `HW1.sml`. The problem solution will consist of a sequence of function definitions in one file. Note that this is a plain text file which you can create and 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. Please 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).

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 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 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. 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.

- 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` – 10%

a) **[8pts]** 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`

2. `listUnion` - 15%

Write a function `listUnion` that takes two lists as input and returns the union of those lists. Your function should have type `''a list -> ''a list -> ''a list`.
Each value should appear in the output list only once, but the order does not matter. Please note that the input lists may have duplicate values or there may be values that appear in both input lists. All such duplicate values should be removed.

Examples:

```
> listUnion [1,3,4] [2,3,4,5]
[1,2,3,4,5]
> listUnion [1,1,2,3,3,3] [1,3,4,5]
[2,1,3,4,5]
> listUnion ["a","b","c"] ["b","b","d"]
["c","a","b","d"]
> listUnion [[1,2],[2,3]] [[1],[2,3],[2,3]]
[[1,2],[1],[2,3]]
```

3. `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 its **`n`**th element is **`v`**. Assume 0-based indexing for **`n`** and **`n`**≥0 . (Note that **`n`** can be greater 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]
> replace 5 6 [1,2,3,4,5]
[1,2,3,4,5]
> replace 6 7 [1,2,3,4,5]
[1,2,3,4,5]
```

4. prereqFor – 20%

Assume that we store the list of CptS courses and their prerequisites as a list of tuples. The first element of each tuple is the course name and the second element is the list of the prerequisites for that course. See below for an example. Please note that a course may have an arbitrary number of prerequisites.

```
val prereqsList = [
("Cpts122" , ["CptS121"]),
("CptS132" , ["CptS131"]),
("CptS223" , ["CptS122", "MATH216"]),
("CptS233" , ["CptS132", "MATH216"]),
("CptS260" , ["CptS223", "CptS233"]),
("CptS315" , ["CptS223", "CptS233"]),
("CptS317" , ["CptS122", "CptS132", "MATH216"]),
("CptS321" , ["CptS223", "CptS233"]),
("CptS322" , ["CptS223","CptS233"]),
("CptS350" , ["CptS223","CptS233", "CptS317"]),
("CptS355" , ["CptS223"]),
("CptS360" , ["CptS223","CptS260"]),
("CptS370" , ["CptS233","CptS260"]),
("CptS427" , ["CptS223","CptS360", "CptS370", "MATH216", "EE234"])
]
```

a) **[18pts]** Assume that you are creating an application for WSU. You would like to write an ML function `prereqFor` that takes the list of courses (similar to above) and a particular course number (in tuple form) and returns the list of the courses which require this course as a prerequisite.
The type of `prereqFor` should be `('a * ''b list) list * ''b -> 'a list.`
(Hint: You can make use of `exists` function you defined earlier.)

Examples:

```
> prereqFor (prereqsList,"CptS260")
["CptS360","CptS370"]
> prereqFor (prereqsList,"CptS223")
["CptS260","CptS315","CptS321","CptS322","CptS350","CptS355","CptS360",
 "CptS427"]
> prereqFor (prereqsList,"CptS355")
[]
```

```
> prereqFor (prereqsList,"MATH216")
["CptS223","CptS233","CptS317","CptS427"]
```

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

5. `isPalindrome` – 20%

A palindrome is a sentence or phrase that is the same forwards and backwards, ignoring spaces, punctuation and other special characters, and upper vs. lower case. In this problem we will consider palindromes that include only letters, digits, and spaces but don't include any punctuation or any special characters — for example "a01 02 2010A", "Yreka Bakery", and "Doc note I dissent a fast never prevents a fatness I diet on cod". Assume that letters are case insensitive — for example "Madam Im Adam" Write an ML function `isPalindrome` that takes a string as argument and that returns `true` if the string is a palindrome and `false` otherwise. (Note: you can make use of the following ML functions: `rev` (for reversing a list), `String.explode` (for retrieving the list of characters in a given string), and `Char.toUpper` (for retrieving the uppercase of a given character).)

The type of `isPalindrome` should be: `string -> bool`.

Examples:

```
> isPalindrome "a01 02 2010A"
true
> isPalindrome "Doc note I dissent a fast never prevents a fatness I diet on
cod"
true
> isPalindrome "Yreka Bakery"
true
> isPalindrome "top cart pop tracPOT"
```

6. `groupSumtoN` – 20%

`groupSumtoN` function takes two arguments where the first argument is an integer (`N`) and the second is a list (`L`). The goal is to produce a result in which the elements of the original list have been collected into ordered sub-lists each containing maximum number of consecutive elements from `L` summing up to `N` or less (where `N` is the integer argument). The leftover elements (if there is any) are included as the last sub-list with a sum less than `N`. If an element in the input list `L` is greater than `N`, that element should be included in its own sublist (including that element only).

The type of `groupSumtoN` should be: `int -> int list -> int list list`.
Note: this function is not required to be tail-recursive.

Examples:

```
> groupSumtoN 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[[1,2,3,4,5],[6,7],[8],[9],[10]]
> groupSumtoN 11 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[[1,2,3,4],[5,6],[7],[8],[9],[10]]
```

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

Testing your functions

For each problem, write a test function that compares the actual output with the expected (correct) output.

- For `exists`, `replace` and `prereqFor`, the instructor will provide the test functions. But you should add additional test inputs for these functions testing the uncovered boundary cases.
- For `listUnion`, `isPalindrome`, and `groupSumtoN`, you need to provide your own test functions. Again, your test inputs should cover the boundary cases as much as possible.

If you don't change the test inputs or don't provide the additional test functions your will be deduced at least 5% in this homework. Test functions should be included at the end of your HW1.sml file.

Below is an example test function for `exists`:

```
fun existsTest () =
   let
     val existsT1 = (exists(8,[7]) = false )
     val existsT2 = (exists("one",["two","one"]) = true )
     val existsT3 = (exists(true,[false,false]) = false )
   in
     print ("\n------------- \nexists:\n" ^
            "test1: " ^ Bool.toString(existsT1) ^ "\n" ^
            "test2: " ^ Bool.toString(existsT2) ^ "\n" ^
             "test3: " ^ Bool.toString(existsT3) ^ "\n")
   end
val _ = existsTest()
```

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:

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