Programming Languages
Assignment IV - Programming project
To be handed in no later than: 21.03.2017 23:59

This assignment consists of four exercises. You should write the solution to the first two in Scheme (50%), and the latter two in SML (50%). One or more test cases are attached to each exercise, for you to use, but additionally, you need to test your functions using Mooshak. Note that you are not allowed to use built-in functions (except base functions like head and tail) to solve the given problem, unless it is stated explicitly. The only exception is that you can use higher order functions like map as you wish (except for 4. e)).

1. **Scheme** (28%)

   a) (5%) Write the function `maxElement` which takes a list `lis` (of length ≥ 1) as an argument. The function returns the largest element of `lis`. You can use the built-in function max in your solution.

   Test case:
   ```
   (maxElement '(5 3 7 2 6 1 4))
   7
   ```

   b) (5%) Write the function `elementAtIndex` which takes a list `lis` (of length ≥ 1) and a number `n` ≥ 1 as arguments. The function returns the n-th element from `lis` (the first element is number 1).

   Test case:
   ```
   (elementAtIndex '(a b c d e f) 4)
   d
   ```

   c) (5%) Write the function `compress` which accepts a list `lis` as an argument and returns a new list in which repeated consecutive elements are replaced with a single copy of the element.

   Test case:
   ```
   (compress '(a a a b b c c c d d d d e f f))
   '(a b c d e f)
   ```

   d) (5%) Write the function `maxofmax` which accepts a list of lists as an argument. The function returns the maximum element in all the lists. You **need** to use the `map` function and `maxElement` (from a)) in your solution.

   Test case:
   ```
   (maxofmax '((5 3 6 2) (1 6 2 7) (7 3 8 2 9) (6 2 4 1)))
   9
   ```

   e) (8%) Write the function `split` which splits an argument list into two lists where the length of the first list is given as the second argument.
   Hint: Use the `list` function and two helper functions.

   Test case:
   ```
   (split '(a b c d e f g) 3)
   '((a b c) (d e f g))
   ```

2. **Scheme** (22%)

Assume that a document is represented in Scheme as a list of lists of symbols.
Each inner list then denotes a sentence, i.e. a list of words.

Example document:

```
'((Hello Mary) (Scheme is so wonderful) (See you later) (John))
```

a) (4%) Write the function `sumList` that returns the sum of the elments in the argument list.

Test case:
```
(sumList '(5 4 3))
12
```

b) (4%) Write the function `charactersPerWord` that accepts a sentence (list of words) as an argument and returns a list containing the length of each word in the sentence. Use `map` and the following function in your solution:

```
(define (charCount sym) (string-length (symbol->string sym)))
```

Test case:
```
(charactersPerWord ' (Scheme is so wonderful))
'(6 2 2 9)
```

c) (6%) Write the function `charactersPerSentence` that accepts a list of sentences (a document) as an argument and returns a list containing the length of each sentence (not counting spaces). Use `map` and the functions from a) and b) in your solution.

Test case:
```
(charactersPerSentence '((Hello Mary) (Scheme is so wonderful) (See you later)
(John)))
'(9 19 11 4)
```

d) (4%) Write the function `wordsPerSentence` that accepts a list of sentences (a document) and returns a list containing the word count of each sentence. Use `map` and the built-in function `length`.

Test case:
```
(charactersPerSentence '((Hello Mary) (Scheme is so wonderful) (See you later)
(John)))
'(2 4 3 1)
```

e) (4%) Write the function `statsDocument` that accepts a document (a list of sentences) and returns a list of three elements. The first element is the character count (not including spaces) in the document, the second element is the word count, and the third element is the sentence count.

Test case:
```
(charactersPerSentence '((Hello Mary) (Scheme is so wonderful) (See you later)
(John)))
'(43 10 4)
```

3. **SML** (25%)

a) (5%) Write the function `zip = fn : 'a list -> 'b list -> ('a * 'b)` list which pairs up the corresponding elements of the two argument lists. If one list is longer than the other, extra elements of the longer list are ignored.

Test case:

```
zip [1,2,3] ["a","b","c"];
val it = [(1,"a"),(2,"b"),(3,"c")] :  (int * string) list
zip [1,2] ["a"];
val it = [(1,"a")] :  (int * string) list
```

b) (5%) Write the function `greaterThan = fn :  int list -> int -> int list` which takes a list `lis` as its first argument and a number `k` as its second argument. The function returns a new list which contains the elements from `lis` which are greater than `k`.

Test case:

```
greaterThan [1,5,3,2,4] 3;
val it = [5,4] :  int list
```

c) (7%) Write the function `reduction = fn:  ('a * 'a -> 'a) -> 'a list -> 'a` which accepts two arguments, function `f` and list `lis` (of length ≥ 1). The function `reduction` replaces pairs of values (x1,x2) from `lis` with the value produced by applying the function `f` to x1 and x2 until there is only one value left in the list – `reduction` returns this value.

Here is how the successive stages in the reduction might look when `f` is `+` and `lis` is `[3,7,-2,5]`.

`[3,7,-2,5] -> [10,-2,5] -> [8,5] -> 13`

Test case:

```
reduction op+ [1,3,5,7,9];
val it = 25 :  int
reduction op* [1,3,5,7,9];
val it = 945 :  int
```

d) (8%) Write the function `partition = fn :  ('a -> bool) -> 'a list -> 'a list * 'a list` which takes two arguments, a function `f` and a list `lis`, and returns a pair of lists. The value of the expression `partition f lis` is the pair `(lis1, lis2)`, where `lis1` contains those elements `e` of `lis1`, where `f(e)` = true, while `lis2` contains the remaining elements of `lis`.

Hint: A **let** statement can help!

Test case:

```
partition Char.isLower [#"P",#"a",#"3",#"%",#"b"];
val it = ([#"a",#"b"],[#"P",#"3",#"%"]) :  char list * char list
```

4. **SML** (25%)

a) (4%) Write the function `insert = fn : real * real list -> real list` which puts the argument `x` into the correct place in the sorted argument list `lis` (ascending order).

Test case:
```
insert (3.3, [1.1, 2.2, 4.4, 5.5]);
val it = [1.1,2.2,3.3,4.4,5.5] :  real list
```

b) (4%) Write the function `insertsort = fn : real list -> real list` which sorts its argument in ascending order. This function uses the `insert` function from a).

Test case:
```
insertsort [2.2, 4.4, 5.5, 3.3, 1.1];
val it = [1.1,2.2,3.3,4.4,5.5] :  real list
```

c) (4%) Write the function `middle = fn : 'a list -> 'a` that returns the middle element of the argument list. If the list has an even number of elements, the function returns the first element of the second half of the list. You can assume that the argument list contains at least one element.

Test case:
```
middle [1,2,3,4,5];
val it = 3 :  int
middle [true,false];
val it = false :  bool
```

d) (5%) Write the function `cartesian = fn : 'a list -> 'b list -> ('a * 'b) list` that takes as input two lists `lis1` and `lis2` and returns a list of pairs that represents the Cartesian product of `lis1` and `lis2`. You can assume that both argument lists contains at least one element. You can use the append (@) operator in your solution.

Test case:
```
cartesian ["a","b","c"] [1,2];
val it = [("a",1),("a",2),("b",1),("b",2),("c",1),("c",2)] :  (string * int) list
```

e) (7%) The built-in function `map` is implemented in curried form, which means that one can partially instantiate the function with, for example:

```
val mapsquare = map (fn x => x * x);
val mapsquare = fn :  int list -> int list
```

Then we can apply for example:

```
mapsquare [1,2,3,4];
val it = [1,4,9,16] :  int list
```

To simulate this behavior, implement the function `mymap = fn : ('a -> 'b) -> 'a list -> 'b list` which takes a single argument, a function f, and returns a function that takes a list `l` and returns a new list that is the result of applying f to each element of `l`.

Test case:
```
(mymap (fn x => x*x)) [1,2,3,4]
```

```
val it = [1,4,9,16] :  int list
```

What to hand in: You need to return two files in MySchool: project2.rkt and project2.sml, for the Scheme/Racket code and SML code, respectively. At the end of the files you should put the test cases shown above. You also need to test your solutions in Mooshak.