Programming Languages and Paradigms
COMP 302, Fall 2016

Assignment 1

Due date: Monday, October 3, 2016
6pm

This assignment focuses on JavaScript and functional programming. Your code must run without error or modification in current versions of Firefox or Chrome: restrict yourself to basic JavaScript idioms defined in the ECMA standard 5.1. Be sure to respect precise naming, input, and output requirements.

All code should be well-commented, in a professional style, with appropriate variables names, indenting (uses spaces and avoid tabs), etc. **The onus is on you to ensure your code is clear and readable. Marks will be very generously deducted for bad style or lack of clarity.**

Your code should endeavour to follow a pure functional programming style. In particular, and unless specifically stated otherwise, **loops are not allowed** (use recursion), and while you may declare and use local variables, **variables may not be modified once assigned or bound**.

When using pairs (lists, trees), base your design on the `cons`, `car`, `cdr`, `isList`, and `show` functions described in class, and given in the *pairs.js* file.

1. We can generalize the `car`, and `cdr` functions to construct a family of accessors, `c...r`, where the "..." is an arbitrary combination of `a`'s and `d`'s. Each `a` means to return the first element, and each `d` means to return the second element of the given pair, read right to left. **7**

   For example, `cadr` returns the car of the cdr of the input pair:

   `cadr(cons(a,(cons(c,(cons(d,null)))))) = c`

   Define a generic constructor `cXXXr` function which can create any of such function. Your code should accept a single parameter, a (possibly empty) string consisting of only `a`'s and `d`'s, and it should return a function which extracts the appropriate element from a cons-list/tree in the way the string specified.

2. Of course we need some interesting lists with which to test the function you build in the above question. The file *q2setup.js* includes code to generate a tree with a random structure. It returns an object containing a non-empty tree (ie the root pair), and the name of an arbitrary element in that tree. **10**

   Write a function `makeXXX(tree,s)`. This function receives a tree and a string (which it certainly contains), and should search the tree for the given string, returning the string you would pass to your `cXXXr` function from the first question to access the corresponding element. That is,

   `cXXXr(makeXXX(tree,s))(tree) == s`

3. The `show` function returns a string describing a recursively defined pair. It would be useful to be also able to go in the other direction too. Define a function `wohs` which accepts a single string parameter and returns the corresponding, recursively constructed `cons` structure. You may assume the list is well-formed in the sense that each list ends in a `()`, and that actual elements are composed of alphanumeric strings. It should be the case that for any tree `t`, **12**

   `show(wohs(show(t))) == show(t)`

4. You are given a list, which you need to partition into a list of lists based on different criteria. For instance, given a list of random integers, you may need to return a list of two lists, the odds and evens. **7**

   Define a function `partition` which takes a list and an arbitrary number of functions as parameters, and returns a list of lists, where each sublist contains only and all elements for which the corresponding function returns true if passed that element. The returned list of lists should respect the order of the input functions, and individual sublists should preserve the original order of elements in the input list.

5. JavaScript uses associative arrays. Show how to build up that functionality from `cons` pairs. Define the following functions. **4**

   - `constructAA()` Returns a new, empty associative array.
   - `addAA(aa,key,value)` Adds the given mapping to the associative array and returns the new associative array. If the key already exists it is overwritten.
   - `getValueAA(aa,key)` Returns the value associated with that key, or null if not found.
   - `showAA(aa)` Returns a formatted string showing the `key:value` mappings in the given associative array. This should be a multiline string with each mapping on a separate line, separating key from value by a colon. For example, the result of
     `showAA(addAA(addAA(constructAA(),"name","clark"),"age",10000))`
     would be:
     ```
     name:clark
     age:10000
     ```

   Note that keys can be arbitrary strings, and values arbitrary data.

What to hand in

Submit your assignment to *MyCourses*. Note that clock accuracy varies, and late assignments will not be accepted without a medical note: **do not wait until the last minute**. Assignments must be submitted on the due date **before 6pm**.

For each question n, include a file `qn.js` with the source code of your answer. You do not need to include the *pairs.js* file, nor the *a2setup.js* file.

This assignment is worth 10% of your final grade. **40**