National University of Singapore
School of Computing
CS1101S: Programming Methodology (JavaScript)
Semester I, 2012/2013

Discussion Group Exercises 4

Problems:

1. Draw box-and-pointer for the values of the following expressions. Also give the printed representation.

```
list(list(1,2,list(3)),list(4,5),pair(6,7))

pair(1,list(2,3,pair(4,[])))

pair(1,pair(2,list(3,list(4,5))))
```

2. Write expressions using `head` and `tail` that will return 1 when the `lst` is bound to the following values:

```
list(7,list(6,5,4),3,list(2,1))

list(list(7),list(6,5,4),list(3,2),1)

list(7,list(6),list(5,list(4)),list(3,list(2,list(1))))

list(7,list(list(6,5),list(4),3,2),list(list(1)))
```

3. Write a JediScript function called `every_second` that takes in a list as its only argument and returns a list containing all the elements of even rank (i.e. every second element) from the input list.

```
every_second(list("a","x","b","y","c","z","d"))
// Value: ["x",["y",["z",[]]]]
```

4. Write a JediScript function called `sums` that takes in a list of integers as its only argument and returns a list of two elements: the first is the sum of all odd-ranked numbers in the input list, whereas the second element is the sum of all even-ranked elements in the input. You may make use of the following accumulate function, provided in the `list.js` library.

```
function accumulate(op,initial,sequence) {
    if (is_empty_list(sequence)) {
        return initial;
    } else {
        return op(head(sequence),
                  accumulate(op,initial,tail(sequence)));
    }
}
```

Example call:

```
sums(1,2,3,4,5)
// Value: [9,[6,[]]]
```

5. This question asks you to build a variant of the solution to the "Towers of Hanoi" problem presented in class. We define a *disk move* to be a list of two numbers: the source pole and the destination pole. For example, `[1,[3,[]]]` indicates the move of a disk from the first pole to the third.

   Write a JavaScript function called `hanoi`, which takes in 4 parameters:

   - the number of disks,
   - the source pole.
   - the destination pole,
   - the auxiliary pole,

   and *returns a list of disk moves* that, if executed in that sequence, will move all the disks from the source pole to the destination pole and comply with the rules of the Tower of Hanoi game. (Hint: you will not get any marks for a solution that *prints* a sequence of moves, since that has already been given in class). You may make use of the following function `append`, provided in the `list.js` library, that appends the second argument list to the first.

```
function append(list1,list2) {
   if (is_empty_list(list1)) {
      return list2;
   } else {
       return pair(head(list1),append(tail(list1),list2));
   }
}
```

   Example call:

```
hanoi(3,1,2,3)
// Value: list(list(1,2),list(1,3),list(2,3),list(1,2),
//             list(3,1),list(3,2),list(1,2))
```

6. Our friend Louis Reasoner has a pocket full of change. He wants to buy a snack that will cost him x cents, and he wants to know all the ways in which he can use his change to make up that amount. Please help him in writing a JavaScript function which takes as parameters the amount `x` and a list of all the coins Louis has in his pocket, and returns a list of lists, such that each sub-list of the result contains a valid combination to make up `x`. A combination may appear more than once, since it may be using different coins of the same denomination.

   So far, Louis managed to come up with the following incomplete solution:

```
function makeup_amount(x,l) {
    if (is_pair(l)) {
        return append(map(...,
                          ...),
                      makeup_amount(x,tail(l)))
```

```
    } else if (x === 0) {
        return list([]);
    } else {
        return [];
    }
}
```

Example call:

```
makeup_amount(22,list(1,10,5,20,1,5,1,50))
// Value: list(list(1,10,5,1,5),list(1,10,5,5,1),list(1,20,1),
              list(1,20,1),list(10,5,1,5,1),list(20,1,1))
```

Note: The sublist `list(1,20,1)` appears twice. Each appearance of the number `1` refers to a different coin.

Help Louis by filling in the ellipses `...` in the given template.

Task Files

- lib/list.js
- **discussion_4_6.js**

7. The function `accumulate_n` is similar to `accumulate` except that it takes as its third argument a sequence of sequences, which are all assumed to have the same number of elements. It applies the designated accumulation function to combine all the first elements of the sequences, all the second elements of the sequences, and so on, and returns a sequence of the results. For instance, if `s` is a sequence containing four sequences, `list(list(1,2,3),list(4,5,6),list(7,8,9),list(10,11,12))`, then the value of

   ```
   accumulate_n(plus, 0, s)
   ```

   should be the sequence `list(22,26,30)`. Fill in the missing expressions in the given definition of `accumulate_n`:

Task Files

- lib/list.js
- **discussion_4_7.js**

8. Disregarding punctuation marks, an English sentence is simply a collection of words, and each word is a sequence of letters. So suppose we'd like to (for no apparent reason) create a list representation for an english sentence, we could simply represent each word as a list of letters (for example, the word 'cat' could be represented as `list("c", "a", "t")`. A sentence could then simply be a list of such words. Some examples:

   ```
   // "I am rich"
   list(list("I"), list("a", "m"), list("r", "i", "c", "h"));

   // "JediScript is cool"
   list(list("J","e","d","i","S","c","r","i","p","t"),
        list("i","s"),
        list("c","o","o","l")
       )
   ```

Task Files

- lib/list.js
- **discussion_4_8.js**

(a) Write a function `count_in_sentence` that takes a `list_sentence` (as described above) and returns a list with two elements: the number of words in the sentence, and the number of characters in the sentence. Assume that spaces count as one character per space, and that there is exactly one space between each word (but none at the start or end of the sentence). There is one period character at the end of each sentence.

Example call:

```
count_in_sentence(list(list("I"),
                       list("l","i","k","e"),
                       list("p","i","e")))
// returns list(3, 11)
```

(b) Write a function `most_frequent_letters` that takes a list-sentence and returns a list of letters that occur most frequently in the given sentence. If only one such letter exists, then return a list with one element. When counting, treat upper and lower case letters to be equal, and only return lower case letters, using the JavaScript function `String.toLowerCase`.

Example call:

```
most_frequent_letters(list(list("S","h","e"),
                           list("l","i","k","e","s"),
                           list("p","i","e","s")))
// returns list("e","s")
```

(c) Write a function `find_end_with` that takes a list-sentence and an additional list of letters. The procedure should return a list of words from the given sentence that end with the given sequence of letters.

Note: `list("r","o","o","t")` ends with `list("t")`, `list("o","t")`, `list("o","o","t")` and
`list("r","o","o","t")`, but not with `list("t","o")`.

Example call:

```
find_end_with(list(list("I"), list("a","m"), list("s","a","m")))
// returns list(list("a","m"), list("s","a","m"))
```

(d) Pig Latin `http://en.wikipedia.org/wiki/Pig_Latin` is an English language game where ordinary words are 'encoded' into peculiar, foreign sounding equivalents. Here are some example words and their Pig Latin versions :

- apple - appleway
- noodles - oodlesnay
- programming - ogrammingpray

While variants of the rules of encoding exist, we will adopt a simple version as follows.

For words beginning with a vowel, add 'way' at the end of the word. For words beginning with consonant(s), bring all consonants at the front of the word to the back, then add 'ay'.

Write a procedure that takes a list-sentence and returns its pig-latin equivalent. Example call:

```
pig_latin(list(list("t","e","a"),
               list("i","s"),
               list("n","i","c","e")))

// returns list(list("e","a","t","a","y"),
                list("i","s","w","a","y"),
                list("i","c","e","n","a","y"))
```