due Wednesday Nov 2 at midnight (on SacCT) - [2 weeks]  
=============================================================  
  
Write the following Scheme functions:  
  
A. "digitinc4" - takes as input a 4-digit integer, and returns   
 another 4-digit integer constructed of the original input  
 integer's digits, each incremented. For example, (digitinc4 4833)  
 would return 5944. When a digit is a 9, the corresponding output  
 digit should be a 0. For example, (digitinc4 4892) would  
 return 5903. You can assume that the input number is at least  
 1000 - that is, there are no leading zeros.  
  
 (\*\*\* HINT: note that simply adding "1111" is not a correct solution,  
 because when one of the numbers is a "9", it would result in a carry,  
 and the digit to its left would get incremented twice.)  
  
B. Extend your answer in problem (A), above, to handle input integers  
 with an arbitrary number of digits. This function should be named  
 "digitinc" (without the "4" in the name). For example, (digitinc 83)  
 would return 94, and (digitinc 22897) would return 33908.  
  
C. "listPicker" - takes as input two lists. The first list is the data,  
 and the second list is the picker. The picker list consists only  
 of integers ranging from 1 to the length of the data list.  
 The output is another list consisting of the values in the data  
 list referenced by the values in the picker list. For example,  
 (listPicker '(42 3 "hello" 99 "bye" 7) '(4 3 3)) would return  
 (99 "hello" "hello") because those are the 4th, 3rd, and 3rd  
 items in the data list.  
  
D. "neshtlist" - takes a list and returns the "neshted" version of  
 the list. "Neshting" a list (ok, your instructor invented it)  
 means surrounding the rightmost and leftmost elements in parens,  
 and repeating the process inwards until the entire list is "neshted".  
 For example, (neshtlist '(4 5 3 2 8)) would return (4 (5 (3) 2) 8).  
 You can assume that the input list is "flat", meaning that it  
 doesn't contain any lists. If the input list has an odd number of  
 elements, the center element should be in a list by itself, as  
 shown in the example above. If the input list has an even number  
 of elements, then there should be an empty list in the middle.  
 For example, (neshtlist '(4 5 2 8) would return (4 (5 () 2) 8)).  
  
E. "repChildren" - takes in four parameters:  
 - a binary tree T  
 - a key value K  
 - a new left child value L  
 - a new right child value R  
 The input tree is a list of the form (root leftchild rightchild), where  
 the root is an integer, and the left and right children are trees.  
 An empty tree is denoted by the empty list, and leaf nodes are  
 designated with an empty list. The tree is not necessarily ordered.  
  
 Your function returns a list similar to T, but at each occurrance of K,  
 its children are replaced by L and R. For example:  
 (repChildren '(7 (3 () ()) (6 (5 () ()) ())) 6 1 2)  
 should return:  
 (7 (3 () ()) (6 (1 () ()) (2 () ())))  
  
 Note that the children of "6" are now "1" and "2".  
 Note also that the inserted children are inserted as trees, meaning  
 they must be given left and right children that are empty lists  
 (as shown in the above example).  
  
F. "functionMajority" - takes a boolean function F and a list L, and  
 returns #t or #f, depending on whether F returns more "true" or  
 "false" answers for the values in the list. For example,  
 (functionMajority isOdd '(4 8 3 6)) would return #f, because  
 "isOdd" returns true for one value in the list (3), and false  
 for three values (4, 8, and 6). Since three is larger then one,  
 the majority of answers are false, and so the function returns #f.  
 In the case of a tie, "functionMajority" should return #t.  
  
G. A function "makeMangler", that takes as input a list M of three  
 numbers. It then builds and returns a "Mangler" function based on M.  
  
 The "Mangler" function that is produced (by your function makeMangler)  
 would have the property that it takes as input a list, and returns  
 the "mangled" version of that list. "Mangling" a list means doing  
 the following sequence of operations to each item in a list:  
  
 (1) multiply by the first element in M, then  
 (2) add the second element in M, then  
 (3) multiply by the third element in M.  
  
 For example, if 'makeMangler" was called as follows:  
  
 (makeMangler '(2 3 8))  
  
 a function would be produced that takes as input a list and returns  
 the "mangled" version, based on the list '(2 3 8).  
  
 For example, if the original call had been made as follows:  
  
 (define C (makeMangler '(2 3 8)))  
  
 then the produced function C would behave as follows:  
  
 (C '(4 8 2 9)) \*\*\* would return (88 152 56 168)  
 (C '(-2 3)) \*\*\* would return (-8 72)  
  
 Your task is just to write makeMangler, not "C".  
 Of course, makeMangler should work for ANY input list, not just (2 3 8).  
  
  
=============================================================  
Submission instructions:  
  
- Place all of your Scheme functions in ONE file.  
- Make sure your functions work in Dr-Racket before submitting the file.  
- Include comments in your code clearly delineating each of the problems.