

Project 1

Due Sept. 22, 2008

This project is an exercise of Lisp programming. You are asked to write several Lisp functions for some simple tasks, and apply your code to given test data.

1. **Build A List.** An element of the list consists of its name (a string) and value (an integer), represented as a pair (name value), i.e., it is itself a list. Then a list of elements is represented as a list of lists. Write a Lisp program to insert element x into list L in such a way that the resultant list remains in the ascending order of their elements' values. For simplicity, you can assume that no two elements will have the same name, and elements with different names always have different values.

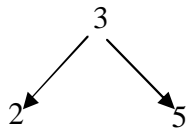
Test data:

Initial list: $L = ((\text{"a"} 2) (\text{"d"} 3) (\text{"c"} 10));$

Elements to be inserted: 1) $(\text{"b"} 5)$ 2) $(\text{"e"} 20)$ 3) $(\text{"f"} 1)$ 4) $(\text{"g"} 6)$

Output: The final list after inserting the above four new elements.

2. **Build A Binary Search Tree (BST).** For simplicity, nodes in the tree are integers, and no duplicate integers will enter the tree. A non-empty BST can be represented by a list (root L1 L2), where root node is an integer, and L1 and L2 are two subtrees whose nodes are less and great than the root, respectively. For example, a tree



can be represented by list $(3 (2 \text{ nil nil}) (5 \text{ nil nil}))$ or $(3 (2 () ()) (5 () ()))$.

Write a Lisp program to construct a BST by successively inserting every integer from a list into the tree, which is initially empty.

Test data: Construct a BST from the following list of integers:

$(10 5 8 28 3 15 6 13 18 21)$

Output: The list that represents the BNS after inserting all of the integers.

3. **Manipulating Complex Lists.** Write the following Lisp functions that deal with complex, nested lists.
 - 1) Generalized membership function *mem-g*. This function takes two arguments S1 and S2, both of which are S-expressions, and returns T if S1 appears anywhere as an element in S2, and NIL otherwise. For example:

$(\text{mem-g 'a '(b (c a)))}$ returns T

$(\text{mem-g '(b c) '(b (c a)))}$ returns NIL

(mem-g '(c a) '(b (c a d))) returns NIL

Apply *mem-g* to the following test data

- a) S1 = a, S2 = ((b (c a) d) x (y (u v)))
- b) S1 = (u v), S2 = ((b (c a) d) (y (u v)))
- c) S1 = (b c a), S2 = ((b (c a) d) (w (u v)))

- 2) Function *replace*. This function takes three arguments S1, S2, and S3 and replaces every occurrence of S1 in S2 by S3. (S2 remains unchanged if S1 does not appear in S1.)

Apply *replace* to the following test data

- a) S1 = a, S2 = Nil, S3 = (x)
- b) S1 = c, S2 = (((a (b c)) c)) S3 = (a b)
- c) S1 = (a b), S2 = (a (b (a b) c) (a b)) S3 = (x y)

4. **Generate Fibonacci Numbers.** Fibonacci numbers are defined recursively as follows:

$$Fib(n) = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ Fib(n-1) + Fib(n-2) & \text{if } n > 1. \end{cases}$$

Write an *efficient* Lisp function *fib(n)* that generates the sequence of Fibonacci numbers Fib(0),...Fib(n).

Test your code with **n = 1**, **n = 5**, and **n = 10**.

Submit. You should submit the following in hard copy:

1. A cover page with your name, class number, date, and a brief summary of what this project is about. You may also write comments and any points you wish to make.
2. Lisp code you wrote.
3. The computer printout of the running result of your code with the test data. The output must be in format that can be easily read and understood.