CS 4100
More fun with lisp
April 27, 2011

==========
Create a file called binarytree.lisp
with the contents below:
==========

```
;;
;; Binary Trees
;;

;;
;; Constructors for binary trees
;;

(defun make-bin-tree-leaf (E)
  "Create a leaf."
  (list E))

(defun make-bin-tree-node (E B1 B2)
  "Create a node with element K, left subtree B1 and right subtree B2."
  (list E B1 B2))

;;
;; Selectors for binary trees
;;

(defun bin-tree-leaf-element (L)
  "Retrieve the element of a leaf L."
  (first L))

(defun bin-tree-node-element (N)
  "Retrieve the element of a node N."
  (first N))

(defun bin-tree-node-left (N)
  "Retrieve the left subtree of a node N."
  (second N))

(defun bin-tree-node-right (N)
  "Retrieve the right subtree of a node N."
  (third N))

;;
;; Recognizers for binary trees
;;

(defun bin-tree-leaf-p (B)
  "Test if binary tree B is a leaf."
  (and (listp B) (= (list-length B) 1)))

(defun bin-tree-node-p (B)
  "Test if binary tree B is a node."
  (and (listp B) (= (list-length B) 3)))
```

==========
Load your file and test the following functions:
==========

```
[1]> (load 'binarytree.lisp)
;; Loading file binarytree.lisp ...
```

```
;; Loaded file binarytree.lisp
T


[2]> (make-bin-tree-node '*
                            (make-bin-tree-node '+
                                                (make-bin-tree-leaf 2)
                                                (make-bin-tree-leaf 3))
                            (make-bin-tree-node '-
                                                (make-bin-tree-leaf 7)
                                                (make-bin-tree-leaf 8)))
(* (+ (2) (3)) (- (7) (8)))


[3]> (defun bin-tree-member-p (B E)
  "Test if E is an element in binary tree B."
  (if (bin-tree-leaf-p B)
      (equal E (bin-tree-leaf-element B))
    (let
     ((elmt  (bin-tree-node-element B))
      (left  (bin-tree-node-left    B))
      (right (bin-tree-node-right   B)))
      (or (equal E elmt)
       (bin-tree-member-p left E)
       (bin-tree-member-p right E)))))
BIN-TREE-MEMBER-P


[4]> (trace bin-tree-member-p)
;; Tracing function BIN-TREE-MEMBER-P.
(BIN-TREE-MEMBER-P)
[5]> (bin-tree-member-p '(+ (* (2) (3)) (- (7) (8))) 7)
T


[6]> (defun binary-tree-reverse (B)
  "Reverse binary tree B."
  (if (bin-tree-leaf-p B)
      B
    (let
     ((elmt  (bin-tree-node-element B))
      (left  (bin-tree-node-left    B))
      (right (bin-tree-node-right   B)))
      (make-bin-tree-node elmt
                  (binary-tree-reverse right)
                  (binary-tree-reverse left)))))


[9]> (trace binary-tree-reverse)
;; Tracing function BINARY-TREE-REVERSE.
(BINARY-TREE-REVERSE)
[10]> (binary-tree-reverse '(* (+ (2) (3)) (- (7) (8))))
(* (- (8) (7)) (+ (3) (2)))


[11]> (defun bin-tree-preorder (B)
  "Create a list containing keys of B in preorder."
  (if (bin-tree-leaf-p B)
      (list (bin-tree-leaf-element B))
    (let
     ((elmt  (bin-tree-node-element B))
      (left  (bin-tree-node-left    B))
      (right (bin-tree-node-right   B)))
      (cons elmt
```

```
         (append (bin-tree-preorder left)
              (bin-tree-preorder right))))))
BIN-TREE-PREORDER


[12]> (trace bin-tree-preorder)
;; Tracing function BIN-TREE-PREORDER.
(BIN-TREE-PREORDER)
[13]> (bin-tree-preorder '(* (+ (2) (3)) (- (7) (8))))
(* + 2 3 - 7 8)

================================================================================
Now write and test inorder and postorder traversal functions
**** You may find it easier to use the modified function and use tree below *****
================================================================================
[17]> (defun bin-tree-preorder (B)
  "Create a list containing keys of B in preorder."
  (if (bin-tree-leaf-p B)
      (list (bin-tree-leaf-element B))
    (let
     ((elmt  (bin-tree-node-element B))
      (left  (bin-tree-node-left    B))
      (right (bin-tree-node-right   B)))
      (append elmt
         (append (bin-tree-preorder left)
              (bin-tree-preorder right))))))
BIN-TREE-PREORDER

[18]> (bin-tree-preorder '((1) ((2) (4) (5)) ((3) (6) (7))))
(1 2 4 5 3 6 7)
```