Patrick Austin  
CS 326

Homework # 2

9/23/2015

;Problem 1

;returns #t if list L contains member x and #f if not. Supporting function used

;throughout problem 1

( define ( member? x L )

( cond

;if empty list reached, x was not found, return #f

( ( null? L ) #f )

;if item matching x found in L, return #t

( ( equal? x ( car L ) ) #t )

( else

;otherwise, search again in cdr of L

( member? x ( cdr L ) ) ) ) )

;a.

;returns T if list L is a set and F if not. Uses member? function

( define ( is-set? L )

( cond

;if empty list reached, no duplicates found, return #

( ( null? L ) #t )

( else

;if first item of L exists in rest of L, there is a duplicate, return #f

( if ( member? ( car L ) ( cdr L ) ) #f

;otherwise, check the remainder of the list for duplicates

( is-set? ( cdr L ) ) ) ) ) )

;b.

;returns a set made from list L. Uses is-set? and member? functions

( define ( make-set L )

( cond

;if the list is a set, just return the list

( ( is-set? L ) L )

( else

;if the first item in L has a duplicate elsewhere in L,

;repeat make-set without it

( if ( member? ( car L ) ( cdr L ) ) ( make-set ( cdr L ) )

;otherwise, cons the first item and the make-set of the

;remaining items together

( cons ( car L ) ( make-set ( cdr L ) ) ) ) ) ) )

;c.

;returns #t if set A is a subset of set S and #f if not.

;Uses member? function.

;Assumes A and S are both valid sets

( define ( subset? A S )

( cond

;if the end of A is reached, it is a subset of S, return #t

( ( null? A ) #t )

( else

;if car of A is contained in S, see if cdr of A is a subset of S

( if ( member? ( car A ) S ) ( subset? ( cdr A ) S )

;otherwise, an item in A was not in S, so A is not a subset of S, return #f

#f ) ) ) )

;d.

;returns a set which is the union of sets A and B.

;Uses member? function.

;Assumes A and B are both valid sets

( define ( union A B )

( cond

;if the end of A is reached, return B

( ( null? A ) B )

( else

;if B already contains car of A, don't include it in B again- repeat

;union operation for cdr of A

( if ( member? ( car A ) B ) ( union ( cdr A ) B )

;otherwise, add car of A to B and repeat union operation for cdr of A

( cons ( car A ) ( union ( cdr A ) B ) ) ) ) ) )

;e.

;returns a set which is the intersection of sets A and B.

;Uses member? function.

;Assumes A and B are both valid sets

( define ( intersection A B )

( cond

;if A is the null set, the intersection is the null set -

;return the null set

( ( null? A ) '() )

( else

;if car of A is a member of B...

( if ( member? ( car A ) B )

;then return cons of car A with the intersection of cdr A and B

( cons ( car A ) ( intersection ( cdr A ) B ) )

;otherwise, car of A is not a member of B -

;return intersection of cdr A with B

( intersection ( cdr A ) B ) ) ) ) )

;Problem 2

;sample tree T, as specified in prompt:

( define T

'(13

(5

(1 () ())

(8 ()

(9 () ())))

(22

(17 () ())

(25 () ()))))

;auxiliary support functions for problem 2:

;provides root of specified tree

( define ( val T ) ( car T ) )

;provides left subtree of specified tree

( define ( left T ) ( car ( cdr T ) ) )

;provides right subtree of specified tree

( define ( right T ) ( car ( cdr ( cdr T ) ) ) )

;a.

;returns #t if the tree contains value V and #f if not

( define ( tree-member? v T )

( cond

;if bottom of the tree reached, value not found, return #f

( ( null? T ) #f )

( else

;if node reached with specified value, return #t

( if ( equal? v (val T) ) #t

;otherwise, search the left and right subtrees for the value and

;return the OR of the results

( or ( tree-member? v ( left T ) ) ( tree-member? v ( right T ) ) ) ) ) ) )

;b.

;returns list of tree elements corresponding to preorder traversal (root, left, right)

( define ( preorder T )

( cond

;if bottom of tree reached, no value found,

;print T (ie nothing)

( ( null? T ) T )

( else

;otherwise, use append to print the preorder list of this subtree.

;Start with list of root (append needs a list not a value)

( append ( list ( val T ) )

;then recursive preorder printout of left subtree

( preorder ( left T ) )

;then recursive preorder printout of right subtree

( preorder ( right T ) ) ) ) ) )

;c.

;returns list of tree elements corresponding to inorder traversal (left, root, right)

( define ( inorder T )

( cond

;if bottom of tree reached, no value found, print T (ie nothing)

( ( null? T ) T )

( else

;otherwise, use append to print the inorder list.

;Start with recursive inorder printout of left subtree

( append ( inorder ( left T ) )

;then list of root (append needs a list not a value)

( list ( val T ) )

;then recursive inorder printout of right subtree

( inorder ( right T ) ) ) ) ) )

;Problem 3

;deletes value v from list L and any sublists of list L

( define ( deep-delete v L )

( cond

;if null list, return null list

( ( null? L ) L )

;if sublist encountered at car of list, cons deep delete in sublist

;and deep delete in rest of list

( ( list? (car L ) ) ( cons (deep-delete v (car L) ) ( deep-delete v (cdr L) ) ) )

( else

;if target value found at car of list,

;repeat deep delete operation with that value excluded from the list

( if ( equal? v ( car L ) ) ( deep-delete v ( cdr L) )

;otherwise, cons car of list with deep delete operation on cdr of list

(cons ( car L) ( deep-delete v ( cdr L ) ) ) ) ) ) )

;Problem 4

;returns a BST which is BST T with value v inserted into it, as per BST rules.

;Assumes v does not already exist in the tree

( define ( insert-bst v T )

( cond

;if null tree, spot for v has been found-

;return subtree consisting of v and two empty nodes

( (null? T ) (list v '() '() ) )

;if v < root, insert left while returning list of subtrees

( ( < v ( val T ) ) ( list ( val T ) ( insert-bst v ( left T ) ) ( right T ) ) )

( else

;otherwise, v > root, insert right while returning list of subtrees

( list ( val T ) ( left T ) ( insert-bst v ( right T ) ) ) ) ) )