Bilkent University

Department of Computer Engineering

CS 315

Programming Language

Homework 3

Programming in Scheme

14.05.2018

Fuad Aghazada: 21503691

Section 1

Instructor:

- Halil Altay Güvenir

Table of contents

Question 1

Code

```
#|
   Function that generates a set from the list given

   @param list - the list that needs to be 'setify'
|#
(define (setify list)
   (cond ((null? list) list)
         ((isElementOf (car list) (cdr list)) (setify (cdr list)))
         (else (cons (car list) (setify (cdr list))))))

#|
   Function that checks whether the given element is
   a member of the given list

   @param element - element that needs to be checked
   @param list - the list that needs to be checked
|#
(define (isElementOf element list)
    (cond ((null? list) #f)
          ((eqv? element (car list)) #t)
          (else (isElementOf element (cdr list)))))
```

Output

```
1> (load "q1_aghazada_fuad.scm")
"/home/cs/fuad.aghazada/cs315/hw/hw3/q1_aghazada_fuad.scm"
1> (setify '())
()
1> (setify '(a))
(a)
1> (setify '(2 a 1.6))
(2 a 1.6)
1> (setify '(a b c a b c d a d))
(b c a d)
> (setify '(2 a 1.6 2 1.60))
(a 2 1.6)
```

Explanation

In order to implement function *setify*, I implemented a helper function *isElementOf*, which is used to determine whether the given element (as parameter) is in the given list (as parameter) or not. Using this function, function *setify* is called recursively by checking all element in the list starting from the first element of the list. If any element is not the element of the list, it is recursively constructed in a list. Function “eqv?” is used for comparing the elements, because it can also be used for comparing non-numeric values [1]. I could not use “eq?”, because it is said that it could not work properly when the types of the elements are different.

Question 2

Code

```
#|
   Function that joins two lists in an ordered way

   @param list1 - first list
   @param list2 - second list
|#
(define (join list1 list2)
      (cond ((null? list1) list2)
            ((null? list2) list1)
            ((> (car list1) (car list2)) (cons (car list2) (join (cdr list2) list1)))
            ((< (car list1) (car list2)) (cons (car list1) (join (cdr list1) list2)))
            (else (cons (car list2) (join (cdr list2) (cdr list1))))
            ))
```

Output

```
> (join '(1 3 5) '())
(1 3 5)
> (join '() '(5 6 7))
(5 6 7)
> (join '(3 5 9) '(2 4 5 6 8 10))
(2 3 4 5 6 8 9 10)
```

Explanation

Function *join* takes two parameters – 2 sorted lists. Firstly, it checks whether the first list is empty – in this case it returns the second list as a result. Otherwise, checks the same condition for the second list. In the next condition, the function compares the first elements on the lists: if first element of the first list is greater than the first element of the second list, the function constructs a list with first element of the second list and for the rest of the list the function is called recursively for the rest of the second list. The same condition is also checked for condition that the first element of the first list is less that the first element of the second list.

Question 3

Code

```
#|
   Function insert-bst is defined and implemented in order to
   insert an element into a Binary Search tree

   @param item - item needed to be inserted
   @param tree - tree to which the item will be inserted
|#
(define (insert-bst item tree)

   (cond ((null? tree) (list item '() '()))
         ((< item (root tree)) (list (root tree) (insert-bst item (leftSub tree)) (rightSub tree)))
         (else (list (root tree) (leftSub tree) (insert-bst item (rightSub tree))))
))

#|
   Function that returns the root of the tree

   @param tree - tree, whose root will be returned
|#
(define (root tree)
   (car tree))

#|
   Function that returns the right subtree of the tree

   @param tree - tree, whose right subtree will be returned
|#
(define (rightSub tree)
   (car (cdr (cdr tree))))

#|
   Function that returns the left subtree of the tree

   @param tree - tree, whose left subtree will be returned
|#
(define (leftSub tree)
   (car (cdr tree)))
```

Output

```
> (load "q3_aghazada_fuad.scm")
"/home/cs/fuad.aghazada/cs315/hw/hw3/q3_aghazada_fuad.scm"
> (insert-bst 8 '())
(8 () ())
> (insert-bst 5 '(8 () ()))
(8 (5 () ()) ())
> (insert-bst 4 '(8 (5 () ()) ()))
(8 (5 (4 () ()) ()) ())
> (insert-bst 4 (insert-bst 5 (insert-bst 8 '())))
(8 (5 (4 () ()) ()) ())
> (insert-bst 7 (insert-bst 4 (insert-bst 5 (insert-bst 8 '()))))
(8 (5 (4 () ()) (7 () ())) ())
```

Explanation

In order to implement the function - *insert-bst,* I also defined and implemented helper functions for getting components of the tree: root, right subtree and left subtree. These function takes the tree, for which it is needed to find one of the mentioned components, as a parameter, and returns the proper result. *Insert-bst* firstly checks whether the tree passed is empty or not, if so it creates a new *node* by making the inserted element a root for the argument tree with empty right and left subtrees. In the next conditions, it is checked that the element that is wanted to be inserted is less or greater that the root of the argument tree: if it is the first case, a list is created with the root, left subtree – recursively called for the left subtree of the argument tree, and right subtree of the argument tree. Otherwise, a list is created with the root, right subtree of the argument tree and right subtree - recursively called for the right subtree of the argument tree.

References

[1] M. S. Aadit, "What is the difference between eq?, eqv?, equal?, and = in Scheme?," StackOverFlow, 18 July 2013. [Online]. Available: https://stackoverflow.com/questions/16299246/what-is-the-difference-between-eq-eqv-equal-and-in-scheme. [Accessed 11 May 2018].