CS326 – Programming Languages, Concepts and Implementation

Homework 2

(Due September 26)

Submission:

- Submit your code in Canvas as one "hw2.txt" file containing all your functions.
- The file must be able to load and be tested in the interpreter.
    - Consequently, the file must be in a plain text format; do not submit Word, PDF, RTF, JPG or any such types of files.
    - Also make sure that any auxiliary information (such as your name or question numbers) is commented out.

1. (55 pts) Consider an implementation of sets with Scheme lists. A set is an unordered collection of elements, without duplicates.

   (a) (11 pts) Write a recursive function `(is-set? L)`, which determines whether the list `L` is a set. The following examples illustrate the use of this function:

   ```
   > (is-set? '(1 2 5))
   #t
   > (is-set? '(1 5 2 5))
   #f
   ```

   (b) (11 pts) Write a recursive function `(make-set L)`, which returns a set built from list L by removing duplicates, if any. Remember that the order of set elements does not matter. The following example illustrates the use of this function:

   ```
   > (make-set '(4 3 5 3 4 1))
   (5 3 4 1)
   ```

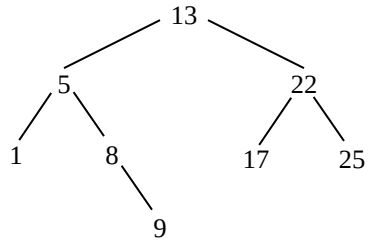
   (c) (11 pts) Write a recursive function `(subset? A S)`, which determines whether the set `A` is a subset of the set `S`.

   (d) (11 pts) Write a recursive function `(union A B)`, which returns the union of sets `A` and `B`.

   (e) (11 pts) Write a recursive function `(intersection A B)`, which returns the intersection of sets `A` and `B`.

2. (33 pts) Consider an implementation of binary trees with Scheme lists, as in the following example:

```
(define T
    '(13
       (5
          (1 () ())
          (8 ()
             (9 () ())))
       (22
            (17 () ())
            (25 () ())))))
```



Before proceeding, it may be useful to define three auxiliary functions `(val T)`, `(left T)` and `(right T)`, which return the value in the root of tree `T`, its left subtree and its right subtree, respectively.

(a) (11 pts) Write a recursive function `(tree-member? V T)`, which determines whether `V` appears as an element in the tree `T`. The following example illustrates the use of this function:

```
> (tree-member? 17 T)
#t
```

(b) (11 pts) Write a recursive function `(preorder T)`, which returns the list of all elements in the tree T corresponding to a preorder traversal of the tree. The following example illustrates the use of this function:

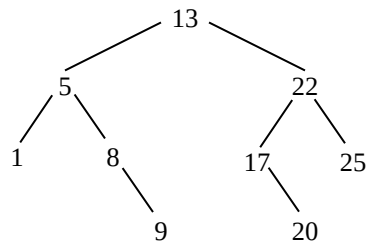
```
> (preorder T)
(13 5 1 8 9 22 17 25)
```

(c) (11 pts) Write a recursive function `(inorder T)`, which returns the list of all elements in the tree T corresponding to an inorder traversal of the tree. The following example illustrates the use of this function:

```
> (inorder T)
(1 5 8 9 13 17 22 25)
```

3. (12 pts) Write a recursive function `(deep-delete V L)`, which takes as arguments a value `V` and a list `L`, and returns a list identical to `L` except that all occurrences of `V` in `L` or in any sublist of `L` have been deleted. The following example illustrates the use of this function:

```
> (deep-delete 3 '(1 2 3 (4 3) 5 (6 (3 7)) 8))
(1 2 (4) 5 (6 (7)) 8)
```

4. (Extra Credit - 10 pts) A *binary search tree* is a binary tree for which the value in each node is greater than or equal to all values in its left subtree, and less than all values in its right

subtree. The binary tree given as example in problem 2 also qualifies as a binary search tree. Using the same list representation, write a recursive function `(insert-bst V T)`, which returns the binary search tree that results by inserting value `V` into binary search tree `T`. For example, by inserting the value 20 into the binary search tree `T` shown above, the result should be: