Spring 1997 **Programming Assignment 2: LISP Warmup** CMSC 420
David Traum

This assignment reflects the philosophy that one learns fastest and best by doing. In this assignment, you will be asked to run LISP programs and observe and describe their behavior.

Part A: Simple Structures

Try out the answers to written assignment 3 on the computer. To begin this exercise type `lisp` into the command shell. At this point, you will be talking to LISP. First type `(dribble "ans2.1")` to have it save the interaction in file ans2.1 in your directory. Now type in the examples in problems 4,6,7, and 8 (remember to use quotes when you need to type in atoms or lists that you don't want lisp to evaluate. Pay attention to what liip returns as a result of these expressions. Now for 9 and 10, use `setq` to assign the proper value of X, Y, and Z, and check your answers to problem 10. Hand in your trace file ans2.1

Part B: Trees

For this exercise, we will need two LISP data structures, `tree1` (a binary tree), and `tree2` (a general tree):

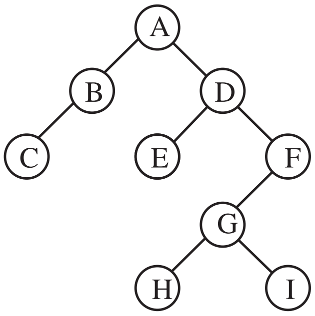

Figure 1: `tree1`

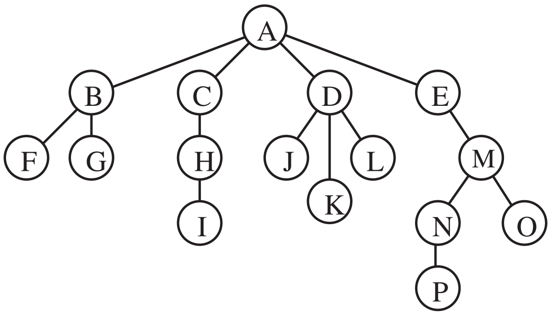

Figure 2: `tree2`

We will represent these two trees in LISP parenthesized notation. To begin this exercise type `lisp` into the command shell. At this point, you will be talking to LISP. First type `(dribble "ans2")` to have it save the output to file ans2 in your directory. Next, set up this exercise by typing the following five s-expressions (alternatively you can copy the file `~dt01/asst2.lisp` to your own directory (before entering lisp, type: `cp ~dt01/asst2.lisp` then inside lisp, type `(load ``asst2.lisp'')`).

Define `tree1`:

```
(setq tree1 '(a (b (c nil nil) nil) (d (e nil nil)
             (f (g (h nil nil) (i nil nil)) nil)))
)
```

Define `tree2`:

```
(setq tree2 '(a (b (f) (g)) (c (h (i))) (d (j) (k) (l))
             (e (m (n (p))(o))))
)
```

Define function `searchb`:

```
(defun searchb (tree node)
    (cond ((null tree) nil)
          (t (or (eq (car tree) node)
                 (searchb (cadr tree) node)
                 (searchb (caddr tree) node)))))
```

Define function `search`:

```
(defun search (tree node)
    (cond ((null tree) nil)
          (t (or (eq node (car tree))
                 (searchsons (cdr tree) node)))))
```

Define functions `searchsons`:

```
(defun searchsons (sonlist node)
    (cond ((null sonlist) nil)
          ((search (car sonlist) node) t)
          (t (searchsons (cdr sonlist) node))))
```

Next, turn on the function trace mechanism for the tree functions so that you may observe the sequence of function calls which are about to occur, as well as the function values:

```
(trace searchb search searchsons)
```

You are now ready to run `searchb` on `tree1` and `search` on `tree2`. Type:

```
(searchb tree1 'g)
```

and pause to observe the trace output. Then do the same for:

```
(searchb tree1 'x)
(search tree2 'k)
(search tree2 'x)
```

When you are finished, exit LISP by typing:

```
(quit)
```

If you ever get yourself into a state you don't understand, type: `:q` until you get to the top level.

Now for the actual problem questions. You may write your written answers on the output listing you have obtained. Turn in both the listing and your written answers.

1. Describe the notation used to represent binary tree `tree1` in LISP parenthesized notation. Write the binary tree in Figure 3 in this notation. Try out your answer by typing it into lisp as tree3 and typing `(searchb tree3 'd)`

2. Describe the notation used to represent the general tree `tree2` in LISP. Write the tree in Figure 4 in this notation. Try out your answer by typing it into lisp as tree4 and typing `(search tree3 'g)`

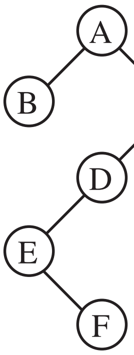

Figure 3: A binary tree

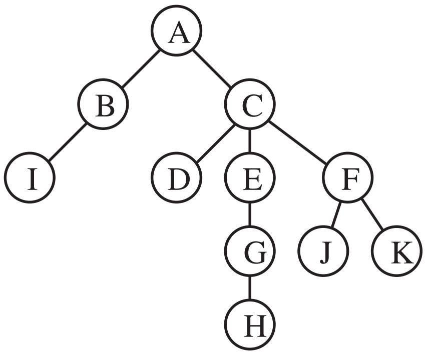

Figure 4: A general tree

3. What does `searchb` do? Identify the recursion relations and the termination test.

4. What does `search` do? Identify the recursion relations and the termination test.

5. What does `searchsons` do? At entry, what kind of data structure is `sonlist`?

6. Why is it necessary to have two distinct recursive functions, `search` and `searchsons`, to process general tree `tree2`, while only one recursive function, `searchb`, is required to process `tree1`? Draw a simple picture illustrating who calls who in the processing of `tree2`.

7. `searchb`, `search` and `searchsons` are really predicates in that they return a logical value, either `t` or `nil`. Suppose we want `searchb` to return a pointer to some part of the binary tree where it now returns only `t`. What would the best value to return be? Write a modified `searchb`, `searchb1`, which would return such a pointer. Run this function and show its output for some examples on tree1 and tree3.