Lisp Programming Lab

Due: Wednesday, February 10th, 1999

This lab is optional. If you are **confident** that you can do the problems here, then you don't have to formally do the lab. We encourage you to try the questions in any case. If you;re an old hand at Lisp, then they should only take a couple on minutes. If you want feedback on your code, bring a hard copy to class on Wednesday and mail a copy to your lab TA. The electronic copy should be valid Lisp code (ie we will load your file into Lispworks and if there are parse errors, then we will send it back to you).

1. Define a function `SUB2` that subtracts two from its input. For example: `(SUB2 10)` should return `8`.

2. Write a predicate `TWOP` that returns `T` only if its input is 2. You should write it in terms of `ZEROP` and `SUB2`.

3. Write a predicate `NOT-ONEP` that returns `T` if its input is anything other than one.

4. Write a function `MY-THIRD` which returns the third item of a list using `FIRST` and `REST`.

5. Write a function called `DUO-CONS` that adds two elements to the front of a list. Remember that the regular `CONS` function adds only one element to a list. `DUO-CONS` would be a function of three inputs. For example, `(DUO-CONS 'PATRICK 'SEYMOUR '(MARVIN)),` should return `(PATRICK SEYMOUR MARVIN).`

6. Define a function `MILES-PER-GALLON` that takes three inputs, `INITIAL-ODOMETER-READING, FINAL-ODOMETER-READING`, and `GALLONS-CONSUMED`, in that order, and computes the number of miles traveled per gallon of gas.

7. The following expressions evaluate without any errors. Write down the result. You should be able to predict the answer before typing them in a lisp interpreter.

```
(cons 5 (list 6 7))
(cons 5 '(list 6 7))
(list 3 'from 9 'gives (- 9 3))
(+ (length '(1 foo 2 moo))
   (third '(1 foo 2 moo)))
(rest `(cons is short for construct))
```

8. The follow expressions all result in errors. Write down the type of error that occurs, explain how the error arose (for example, missing quote, quote in wrong place), and correct the expresssion by changing *only* the quotes.

```
(third (the quick brown fox))
(list 2 and 2 is 4)
(+ 1 '(length (list t t t t)))
(cons 'elliot (daniel bill))
(cons 'ramona (list gort flakey))
```

9. Write a function `CONSTRAIN` that takes a list of three inputs called `X, MIN`, and `MAX`. If `X` is less than or equal `MIN`, it should return `MIN`; if `X` is greater than or equal `MAX`, it should return `MAX`. Otherwise, it should return `X`. For example: `(CONSTRAIN 3 -50 50)` should return 3. `(CONSTRAIN 90 -50 50)` should return `50`. Write one version using `COND` and another using `IF`s.

10. Write a function to act as referee in the Rock-Scissors-Paper game. In this game, each player picks one of Rock, Scissors, or Paper, and then both players tell what they picked. Rock " breaks" Scissors, so if the first player picks Rock and the second picks Scissors, the first player wins. Scissors "cuts" Paper, and Paper "covers" Rock. If both players pick the same thing, it's a tie. The function should take two inputs, each of which is either `ROCK`, `SCISSORS`, or `PAPER`, and return one of `FIRST-WINS`, `SECOND-WINS`, or `TIE`. Examples: `(PLAY 'ROCK 'SCISSORS)` should return `FIRST-WINS`. `(PLAY 'PAPER 'SCISSORS)` should returns `SECOND-WINS`.

11. Write a function `ADD-UP` that takes a list of numbers and adds them all up. You can assume all the element to be numbers. For example: `(ADD-UP '(1 2 3 4))` should return `10`. Do it once without using `FUNCALL` or `APPLY`. Then do it again using `FUNCALL` or `APPLY`.

12. Write `MERGE-LISTS`, a function that takes two lists of numbers, each in ascending order, as input. The function should return a list that is a merger of the elements in its inputs, in order. For example: `(MERGE-LISTS '(1 2 6 8 10 12) '(2 3 5 9 13))` should return `(1 2 2 3 5 6 8 9 10 12 13)`. It should run in time proportional to the length of the resulting list.