Racket Interpreter

**Homework Summary**

For this part of the homework assignment, we will be creating an interpreter for our TINY programming language in scheme. Our program should be able to take an AST (created by our parser) written in a list format, and "execute" those instructions in racket.

**Details**

I have given you a mostly completed racket file that you will need to complete to finish the interpreter. Our program defines an "interpreter" that accepts a "program" (an AST from the previous homework assignment that has been formatted as a list) and executes it. Note that the format of the AST does NOT include the token names (as they did in the assignment).

Should you wish to modify your program so that yours produces the same output, you could then use this to produce more test cases than the ones that I will provide to you. You simply need to modify the toStringList method.

I have included several "programs" at the bottom that you can give your interpreter to execute without having to run your previous assignment to generate programs.

**Referencing Environment**

The reference environment for our TINY language is every variable we've seen the entire time our program has been running. We can't use a variable until it has been created and assigned a value.

When we start our interpreter, one of the first things it will do is create an empty reference environment. It should pass this environment along with a single "statement" (recall we built our AST tree so that a program is a series of statements, where each statement is another child of program). Each time our interpreter encounters an assignment statement, it should add the new variable (and it's value) to the reference environment. Each time a statement is processed, the interpreter should decide whether it's looking at a variable, and if it is, it should search for the value of that variable in the reference environment (that it was passed).

I have provided several of the functions for you:

- empty-env: creates an "empty reference environment"
- extend-env: accepts a reference environment, a variable, and it's value and adds the variable and it's value to the reference environment
- apply-env: accepts a reference environment and a variable and returns its value (or an error message if it cannot find that variable in the reference environment)

**Extra Credit**

**(5 Points)**

Add the ability for your interpreter to handle at least the following Boolean expressions:

<, >, and

**Hint**: in scheme the symbols for the boolean operators above are what is written above (<, >, and)

Racket Interpreter

**(15 Points)**

Add the ability for your interpreter to handle while loops.

**Grading:**

**(20 points each)**

5 test programs similar to [a-e]prog

Some will have errors.

**(5 points extra credit)**

1 test program to see if your interpreter can handle the following boolean operators: **>, <, and**

**(15 points extra credit)**

1 test program to see if your interpreter can handle while loops