CSCI 3155: Lab Assignment 3

2. **JavaScripty Interpreter: Tag Testing, Recursive Functions, and Dynamic Scoping**.

(a) First, write some JAVASCRIPTY programs and execute them as JavaScript programs. This step will inform how you will implement your interpreter and will serve as tests for your interpreter.

Write-up: Give one test case that behaves differently under dynamic scoping versus static scoping (and does not crash). Explain the test case and how they behave differently in your write-up.

One example test case where dynamic scoping provides a different answer than static scoping would be:

```
const x = 10;
const plus = function(x){ return function(y){ return x + y; } };
jsy.print(plus(5)(5));
```

If this was static scoping the result would be: 10

If this were dynamic, the returned the value would be: 15

The static case doesn't take into account the earlier assignment of v1, while the dynamic version will immediately account for v1's value, then use it for the function.

3. **JavaScripty Interpreter: Substitution and Evaluation Order**.

(c) **Write-up**: Explain whether the evaluation order is deterministic as specified by the judgment form $e \rightarrow e_0$.

It is informative to compare the small-step semantics used in this question and the big-step semantics from the previous one. In particular, for all programs where dynamic scoping is not an issue, your interpreters in this question and the previous should behave the same. We have provided the functions evaluate and iterateStep that evaluate "top-level" expressions to a value using your interpreter implementations.

The system is deterministic because it always goes from left to right. The eval function is written to interpret expressions in such a manner. If the same expression were to run multiple times, the results will be the same and consistent.

4. **Evaluation Order**.

Consider the small-step operational semantics for JAVASCRIPTY shown in Figures 7, 8, and 9. What is the evaluation order for $e_1 + e_2$? Explain. How do we change the rules obtain the opposite evaluation order?

For e1 + e2, e1 would come first, then e2, and then +. If it were reversed this we could simply evaluate to e2 being first, then e1, and finally +. This can be done by reversing the eval function. There are three different case methods for adding, and ambiguity is avoided by taking dealing with all possible situations. If the addition is e1 + e2, the first step would be check if e1 or e2 are strings. If e1 is a string, and e2 isn't, use the DoPlusString1 case in figure 7. If e2 is a string and e1 isn't, use the DoPlusString2 case. Otherwise, neither of them are strings, then transform both into numbers and return the sum.

5. **Short-Circuit Evaluation**. In this question, we will discuss some issues with short-circuit evaluation.

(a) **Concept**. Give an example that illustrates the usefulness of short-circuit evaluation. Explain your example.

The "|| =" symbol in Ruby is a fine example of useful short-circuit evaluation. For example "x || = y" will check if x is set to a value, and if so, the evaluation is short-circuited and x is returned; if not, then x is set to the value of y.

(b) **JAVASCRIPTY**. Consider the small-step operational semantics for JAVASCRIPTY shown in Figures 7, 8, and 9. Does e_1 && e_2 short circuit? Explain.

e1 && e2 short-circuits because the program first checks if e1 is valid, if not, false is returned since there isn't a way to proper use the And statement. The And statement cannot be true with one false element.