Kyle Wiese & Sean Donohoe
CSCI 3155
Lab 3 Writeup

1.) 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.

Consider the following javascripty code:

```
const x = 0;

const g  = function p(r) { jsy.print(x) }

const h = function h(y) { const x = 2; g(x) }

h(2)
```

In dynamic scoping, the declaration of x = 2 in function h would mean that function g would print out 2, since x is most recently defined in h at the time we call g. With static scoping, g would print 0, since x is initialized to zero at the scope of g’s declaration.

2.) Explain whether the evaluation order is deterministic as specified by the judgment form e → e’ .

Only when e1 has been fully evaluated to completion do we step on e2, therefore the evaluation order is left associative, thus the evaluation order is deterministic.

3.) Consider the small-step operational semantics for JAVASCRIPTY shown in Figures 7, 8, and 9. What is the evaluation order for e1 + e2? Explain. How do we change the rules obtain the opposite evaluation order?

Using the small step semantics, SearchBinary would be called until e1 is evaluated fully (i.e. we step on e1 until its evaluation), and then SearchBinaryArith would be called until e2 is fully evaluated (i.e. we step on e2 until its evaluation, only after e1 is fully evaluated). On the final call, DoPlusNumber or DoPlusString would be called to evaluate the expression v1 + v2, based on whether or not v1 or v2 are strings. We could change this evaluation order by changing e2 to e1 and vice versa in the SearchBinary and SearchBinaryArith rules. This would mean we step on e2 until it’s fully evaluated, then step on e1 until it’s fully evaluated, and then evaluate the expression v1 + v2, making the evaluation order right associative

4.)

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

```
node* headPointer = linkedList();
if( headPointer != null && headPointer->data == 1)
```

This line checks to see if the headPointer is null before checking the data, which prevents a null pointer exception from being thrown due to the fact that as soon as the first statement is found to be false, the second statement is not evaluated thus preventing an invalid reference to a null pointer.

b. Consider the small-step operational semantics for JAVASCRIPTY shown in Figures 7, 8, and 9. Does e1 && e2 short circuit? Explain.

We call SearchBinary until e1 is fully evaluated. In the case that v1 is equal to true, we would call DoAndTrue and return expression e2. If v1 is false we would call DoAndFalse and return v1. In both cases e2 is not evaluated until the first condition, and is only evaluated if e1 evaluates to true; therefore, it is short circuited.