CSCI 3155: Principles of Programming Languages
Spring 2014: April 8, 2014 **Quiz 5**

Name: _____

We will spend about 15 minutes individually and then 10 minutes with a partner on these questions before discussing as a class.

1. [10 points] **Parameter Passing Semantics**. Let us consider a new kind of parameter passing mode that we describe here. Suppose that we extend imperative JAVASCRIPTY with copy-out parameters where the function must be called with a mutable memory cell (i.e., an l-value). On a call, a new mutable cell is allocated for the parameter initialized to **undefined**. Then, body of the function is executed. Finally, the contents of the parameter cell are written back to the argument cell. We will write such functions with the **out** mode annotation. For this question, assume that all functions have exactly one parameter and are anonymous (i.e., cannot be recursive), so a call-by-result function looks like the following: **function** (**out** $x : \tau$) e_1. For simplicity, assume the type checker rules out reading from x until it has been written to. The function should as usual return the value of e_1. Hint: You will need to step to a new expression with a substitution but not just a substitution.

   (a) Give the "do" rule for calling call-by-result functions.

   Solution:

   DOCALLOUT

   $$\frac{a \notin \mathrm{dom}(M)}{\langle M, (\textbf{function}\ (\textbf{out}\ x : \tau)\ e_1)(lv_2)\rangle \longrightarrow \langle M[a \to \textbf{undefined}], \textbf{const}\ y = [*a/x]e_1;\ lv_2 = *a;\ y\rangle}$$

   For your reference, we give below the "do" rule for calling call-by-value, mutable parameter functions:

   DOCALLVAR

   $$\frac{a \notin \mathrm{dom}(M)}{\langle M, (\textbf{function}\ (\textbf{var}\ x : \tau)\ e_1)(v_2)\rangle \longrightarrow \langle M[a \to v_2], [*a/x]e_1\rangle}$$

Name: _____

(b) Explain briefly how call-by-result differs from call-by-reference (i.e., using a **ref**). Then, give an example program using a **ref** parameter that would be behave differently by changing only that one parameter to a **out** parameter.

Solution: Call-by-result allocates a new local (i.e., at address a in the rule above) and copies the value of the local into the location named by the argument (i.e., lv_2 in the rule above) after executing the function body. In contrast, call-by-reference uses a previously allocated location. Consider the following program:

```
1 var y = 0;
2 (function (ref x: number) { return
3   x = 42,
4   jsy.print(y)
5 })(y);
6 jsy.print(y);
```

In the **ref** case as above, both calls to **jsy**.**print**(y) print 42. However, changing the **ref** parameter x to an **out** parameter will cause the print at line 4 to print 0 even though the print at line 6 will still be 42.

Name: _____

2. [10 points] **Aliasing**. Consider the following program in JavaScript with mutable objects. Ignore the boxed item for the moment.

```
const o = { x: 0, y: 0 };
```

$[a_1 \mapsto \{x:0, y:0\}]$

```
o.x = { f: 1, g: { f: 2, g: 3 } };


o.y = o.x.g;


o.y.g = o.x;


o.y = o.y.g;


o.x = o.x.g;


o.y.f = o.x.f;


jsy.print(o.x.f);
o.y.g.g.f = 7;
jsy.print(o.x.f);
jsy.print(o.y.f);
```

i. To help figure out the values that are printed, annotate the program points with the memory state at each point in the above. The first memory state is given as an example (boxed). Please list and subscript the addresses a based on the order of allocation (in our operational semantics).

ii. What are the three values that are printed?

Name: _____

Solution:

const `o = { x: 0, y: 0 };`

$[a_1 \mapsto \{\mathtt{x}:0, \mathtt{y}:0\}]$

```
o.x = { f: 1, g: { f: 2, g: 3 } };
```

$[a_1 \mapsto \{\mathtt{x}:a_3, \mathtt{y}:0\}, a_2 \mapsto \{\mathtt{f}:2, \mathtt{g}:3\}, a_3 \mapsto \{\mathtt{f}:1, \mathtt{g}:a_2\}]$

```
o.y = o.x.g;
```

$[a_1 \mapsto \{\mathtt{x}:a_3, \mathtt{y}:a_2\}, a_2 \mapsto \{\mathtt{f}:2, \mathtt{g}:3\}, a_3 \mapsto \{\mathtt{f}:1, \mathtt{g}:a_2\}]$

```
o.y.g = o.x;
```

$[a_1 \mapsto \{\mathtt{x}:a_3, \mathtt{y}:a_2\}, a_2 \mapsto \{\mathtt{f}:2, \mathtt{g}:a_3\}, a_3 \mapsto \{\mathtt{f}:1, \mathtt{g}:a_2\}]$

```
o.y = o.y.g;
```

$[a_1 \mapsto \{\mathtt{x}:a_3, \mathtt{y}:a_3\}, a_2 \mapsto \{\mathtt{f}:2, \mathtt{g}:a_3\}, a_3 \mapsto \{\mathtt{f}:1, \mathtt{g}:a_2\}]$

```
o.x = o.x.g;
```

$[a_1 \mapsto \{\mathtt{x}:a_2, \mathtt{y}:a_3\}, a_2 \mapsto \{\mathtt{f}:2, \mathtt{g}:a_3\}, a_3 \mapsto \{\mathtt{f}:1, \mathtt{g}:a_2\}]$

```
o.y.f = o.x.f;
```

$[a_1 \mapsto \{\mathtt{x}:a_2, \mathtt{y}:a_3\}, a_2 \mapsto \{\mathtt{f}:2, \mathtt{g}:a_3\}, a_3 \mapsto \{\mathtt{f}:2, \mathtt{g}:a_2\}]$

```
jsy.print(o.x.f);   // prints 2
o.y.g.g.f = 7;
```

$[a_1 \mapsto \{\mathtt{x}:a_2, \mathtt{y}:a_3\}, a_2 \mapsto \{\mathtt{f}:2, \mathtt{g}:a_3\}, a_3 \mapsto \{\mathtt{f}:7, \mathtt{g}:a_2\}]$

```
jsy.print(o.x.f);   // prints 2
jsy.print(o.y.f);   // prints 7
```