Quiz 4 (2013/10/23)

October 22, 2013

Put your name and student ID clearly on the quiz answer sheet and on a sheet of your own paper. Write "Q1" and "Q2" on your paper. For each question, write your answer on *both* sheets in the appropriate place. Hand in the quiz answer sheet *only*.

In every case, select the single *best* answer.

You're done with any question you answer correctly on the quiz.

By Tue 8PM, submit corrections. Include your name, student ID, the quiz number, and your collaboration statement. (**Even if you got everything right, please do submit for our records**. Just indicate that you got everything correct.) For each question you got incorrect, write the question number and then explain why the correct answer is correct and why the answer you chose is incorrect.

1 Question 1: Eager Evaluation, Static Scoping

Consider the following code:

```
(local [(define x 1)]
  (local [(define y (+ x x))]
    (local [(define x (+ y y))]
      x)))
```

What will the program evaluate to? (Hint: Under eager evaluation with static scoping, everything behaves pretty much the way we expect it to. No surprises!)

```
A. 1

B. 2

C. 4

D. It will "diverge": cause an error or run forever.
```

-

2 Question 2: Lazy Evaluation, Static Scoping

Consider the following code:

```
(local [(define x 1)]
  (local [(define y (+ x x))]
    (local [(define x (+ y y))]
      x)))
```

Under **lazy** evaluation with **static** scoping, identifiers bind to expression closures rather than to values. What will the program evaluate to?

```
A. 1

B. 2

C. 4

D. It will "diverge": cause an error or run forever.
```

-

3 Question 3: Lazy Evaluation, Dynamic Scoping

Consider the following code:

```
(local [(define x 1)]
  (local [(define y (+ x x))]
    (local [(define x (+ y y))]
      x)))
```

Under **lazy** evaluation with **dynamic** scoping, identifiers bind to expressions but **not** expression closures. What will the program evaluate to?

A. 1

B. 2

C. 4

D. It will "diverge": cause an error or run forever.

-

4 Question 4: Closures and Static Scoping

A closure is a data structure representing a function value; closures contain an environment. How does including an environment in a function value enable us to support static scoping?

A. It enables us to use lazy evaluation.

B. It enables us to use eager evaluation.

C. It "remembers" the values bound to the function's parameters.

D. It "remembers" bindings from where the function was defined.

-

5 Question 5: Implementing Function Definition with Closures

When we implement function definition (such as the `FuncC` case of ParseTongue or `lambda` in Racket), we create a closure that stores the environment. Specifically, in our interpreter, we usually store the value of a variable named `env` in the closure.

Where does that variable `env` get its value?

```
A. From the function call site.

B. From the current call to interp.

C. From the initial call to run.

D. From inside the closure itself.
```

•

6 Question 6: Implementing Function Application with Closures

Once we have closures, we no longer need to insist on a function name as the first part of a function application. We can have an identifier bound to a function value (as with `double` in `(double 5)`) or any other expression that evaluates to a function (as with the `(lambda (...) ...)` part of `((lambda (lon) (map double lon)) (list 1 2 3))`).

Which environment should we use when interpreting the function expression to get the function value?

```
A. The environment contained in the closure.

B. The environment stored in the env parameter of interp.

C. The global environment.

D. The empty environment.
```

•