ASSIGNMENT, STATE AND ENVIRONMENTS 8

COMPUTER SCIENCE 61AS

Local State Variables

1. What are local state variables and why do we use them?

Solution: Local state variables are internal variables that are defined when a procedure is defined. They cannot be accessed by any other procedures. We use them for security. For example of the atm machine, if we use local state variable ”balance,” no other procedures can intervene and manipulate the value of ”balance,” which would violate the integrity.

2. What will Scheme output?

```
(define (make-counter number)
  (lambda ()
     (begin (set! number (+ number 1))
            number)))

(define c1 (make-counter 1))
(define c2 (make-counter 1))

(c1)
```

____

```
(c2)
```

____

Solution: 2, 2. Since local state variable, ”number,” of each counter is accessible only by its counter, calling c1 and c2 would not affect each other’s number.

3. At this point, can we still use the substitution model of evaluation? Why or why not?

Solution: No. In substitution model, variable names are replaced by values that they are bound to. However, now we can change the value of variables by set!, so using the substitution model will yield a wrong value. It is better to think of a variable as a place where a value can be stored and that value can change.

4. Fill in the blanks with the values of the expressions shown:

```
(define x 1)
(define foo
  (let ((x x))
    (lambda ()
       (set! x (+ x 1))
       x)))

(foo)                         ___________
(foo)                         ___________
(foo)                         ___________
x                             ___________
```

Solution: 2, 3, 1. The local state variable x in foo is different from x in the global environment. Since x in the procedure foo can never be accessed outside of the procedure, when we call x, we access the value of x in the global environment. x was used for the initial value of the local state variable x, however, it was never modified after that. So x stays as 1.

The Environment Model of Evaluation

1. What is a frame? When are frames created?

Solution: A collection of name-value associations or bindings. They are created when there is a call to a non-primitive procedure.

2. What is the initial frame, if no frame has been created?

Solution: Global frame

3. How is a variable evaluated?

Solution: Find the first available binding. Look in the current frame; if not found there, look in the frame behind the current frame; and so on until the global frame is reached.

4. What are the two things that create a new frame?

Solution: Compound procedure invocation, lambda

Environment Diagrams

Draw environment diagram for each, and say what the code evaluates to.

1.
```
(define (foo x)
(bar x 4))
(define (bar y z)
(- y z))
(foo 9)
```
____

Solution: 5

2.
```
(define x 10)
(define y 20)
(let  ((x y)
  (y x))
(set! x y)
(set! y x)
(- x y))
```
____

```
y
```
____

Solution: 0, 20

3.
```
(define (compose f g)
(lambda (x) (f (g x))))
(define mystery
       (compose
(lambda (x) (+ x 5))
(lambda (x) (* x 10))))
```

```
(mystery 123)
```

____

Solution: 1235

OOP Below the Line

Lets revisit Tic-Tac-Toe! Rewrite the board class without using the OOP language. The code in OOP is given below. After you rewrite it, there will be a different way of instantiating a board, and a different way of calling a method on a board, analogous to the transformation of the counter class shown in the lecture notes/webcast.

Then, give a sample interaction which creates a board, and invokes play-move with piece x, and coordinates (1, 0). Show the environment diagram generated by this interaction.

```
(define-class (board)
  (instance-vars (grid (make-grid)))
  (method (piece x y) (get-piece grid x y))
  (method (play-move piece x y)
    (set! grid (next-grid grid piece x y)))
```

Solution:

```
(define (board)
(let ((grid (make-grid)))
(lambda (msg . args)
(cond ((eq? msg piece)
  (get-piece grid (car args) (cadr args)))
((eq? msg play-move)
  (set! grid (next-grid grid (car args) (cadr args) (caddr args))))
(else (error Bad message))))))
```