Recitation 14

Environment Model

Eval

- *name* - Look up *name* in the current environment, if found return value, otherwise lookup in parent environment.

- *(lambda (params) body)* - Create double bubble with code ptr to *params* and *body* and env ptr to current env.

- *(define name value)* - Evaluate *value* and then create/replace binding for *name* with the result.

- *(set! name value)* - Evaluate *value* and then replace the closest binding for *name* in the chain of environments, starting with the current env.

- *(proc args ... )* - Evaluate *proc* and *args* in the current env, then apply.

- Otherwise - Do The Right Thing (DTRT)

Apply

- Step 1 - Drop a new frame

- Step 2 - Link frame ptr of new frame to env pointed to by env pointer of double bubble being applied.

- Step 3 - Bind params of double bubble in the new frame.

- Step 4 - Eval the *body* in the new frame.
Hats

Break up task into a couple of separate roles:

- **Double-Bubble** - In charge of the lambda rule
- **Bind** - In charge of step 2 apply, define rule, and set! rule
- **Trouble** - In charge of step 3 apply (’cause it’s trouble indeed)
- **Grand Evaluator** - In charge of keeping track of evaluation, current environment, identifying the type of expression, and remembering the values of arguments.

Problems

Problem 1

```scheme
(define square (identity (lambda (x) (* x x))))
(square 5)
```

Problem 2

```scheme
(define (sum-of-squares x y)
  (+ (square x) (square y)))
(sum-of-squares 2 3)
```
Problem 3

(define x 3)
(((lambda (x y) (+ (x 1) y))
 (lambda (z) (+ x 2)))
 3)

Problem 4

(define (fact n)
  (if (= n 0)
   1
   (* n (fact (- n 1)))))
(fact 2)
Problem 5

(let ((x 5)
      (y (+ x 5)))
  (+ x y))

Problem 6

(define (previous f)
  (let ((old false))
    (lambda (x)
      (let ((return old))
        (set! old (f x))
        return)))
(define echo (previous (lambda (y) y)))
(echo 1)