GENERIC OPERATORS AND RACKET1.RKT 6

COMPUTER SCIENCE 61AS

Racket-1 Practice

1. Exercise 1 (Long): Show how racket1 evaluates the following expression. Show all of the calls to eval-1, apply-1 and substitute. Dont show recursive calls to substitute though. (In essence, if we traced all 3 of these procedures, but ignored recursive calls to substitute, what would be the output?) Note: If you want to understand how substitute works, you should trace the recursive calls to substitute as well.

```
(((lambda (x)
     ((lambda (y)
        (lambda (x) (+ x y x)))
     x)) 5) 10
```

2. If I type this into Racket, I get an unbound variable error: (eval-1 'x) Why didnt this just return x, unquoted? What should I have typed in instead? (Assume that I want to use eval-1 from Racket in order to get the symbol x, unquoted.)

3. Hacking Racket-1: For some reason, this expression works:

```
('(lambda (x) (* x x)) 3)
```

In Racket, this would cause an error, because of the quote in front of the lambda expression. Why does it work in Racket-1? What fact about Racket-1 does this exploit?

```
(define (eval-1 exp)
    (cond ((constant? exp) exp)
          ((symbol? exp) (eval exp)) ; use underlying Racket's EVAL
          ((quote-exp? exp) (cadr exp)
          ((if-exp? exp)
              (if (eval-1 (cadr exp))
                  (eval-1 (caddr exp))
                  (eval-1 (cadddr exp))))
          ((lambda-exp? exp) exp)
          ((pair? exp) (apply-1 (eval-1 (car exp)) ; eval the operator
                                (map eval-1 (cdr exp))))
          (else (error "bad expr: " exp))))
(define (apply-1 proc args)
    (cond ((procedure? proc) ; use underlying Racket's APPLY
              (apply proc args))
          ((lambda-exp? proc)
              (eval-1 (substitute (caddr proc) ; the body
                                  (cadr proc) ; the formal parameters
                                  args ; the actual arguments
                                  '()))) ; bound-vars, see below
          (else (error "bad proc: " proc))))
(define (substitute exp params args bound)
    (cond ((constant? exp) exp)
          ((symbol? exp)
              (if (memq exp bound)
                  exp
                  (lookup exp params args)))
          ((quote-exp? exp) exp)
          ((lambda-exp? exp)
              (list 'lambda
                    (cadr exp)
                    (substitute (caddr exp)
                                params
                                args
                                (append bound (cadr exp)))))
                (else (map (lambda (subexp)
                                   (substitute subexp
                                               params
                                               args
                                               bound))
                           exp))))
```

Data Directed Programing

Exercise 4: The TAs have broken out in a cold war; apparently, at the last midterm-grading session, someone ate the last potsticker and refused to admit it. It is near the end of the semester, and Professor Hilfinger really needs to enter the grades. Unfortunately, the TAs represent the grades of their students differently, and refuse to change their representation to someone elses. Professor Hilfinger is far too busy to work with five different sets of procedures and five sets of student data, so for educational purposes, you have been tasked to solve this problem for him. The TAs have agreed to type-tag each student record with their (the TAs) first name, conforming to the following standard:

```
(define type-tag car)
(define content cdr)
```

Its up to you to combine their representation into a single interface for Professor Hilfinger to use.

a. Write a procedure (make-tagged-record ta-name record) that takes in a TAs student record, and type-tags it so its consistent with the type-tag and content selector procedures defined above.

b. A student record consists of two things: a name item and a grade item. Each TA represents a student record differently. Marion uses a list, whose first element is a name item, and the second element the grade item. Jisoo uses a cons pair, whose car is the name item, and the cdr the grade item. Make calls to put and get, and write generic getname and get-grade procedures that take in a tagged student record and return the name or grade items, respectively.

c. Each TA represents names differently. Jisoo uses a cons pair, whose car is the last name and whose cdr is the first. Sam is so cool that a name is just a word of two letters, representing the initials of the student (so George Bush would be gb). Make calls to put and get to prepare the table, then write generic get-first-name and get-lastname procedures that take in a tagged student record and return the first or last name, respectively.

d. Each TA represents grades differently. Marion is lazy, so his grade item is just the total number of points for the student. Sam is more careful, so his grade item is an association list of pairs; each pair represents a grade entry for an assignment, so the car is the name of the assignment, and the cdr the number of points the student got. Make calls to put and get to prepare the table, and write a generic get-total-points procedure that take in a tagged student record and return the total number of points the student has.

e. Now Professor Hilfinger wants you to convert all student records to the format he wants. He has supplied you with his record-constructor, (make-student-record name grade), which takes in a name item and a grade item, and returns a student record in the format Professor Hilfinger likes. He also gave you (make-name first last), which creates a name item, and (make-grade total-points), which takes in the total number of points the student has and creates a grade item. Write a procedure, (convert-to-hilfinger-format records), which takes in a list of student records, and returns a list of student records in Professor Hilfingers format, each record tagged with 'Hilfinger.