Quiz 4a

1. (3 points) **What will Scheme print** in response to the following expressions?

If an expression produces an error message, you may just write ``error''; you don't have to provide the exact text of the message. If the value of an expression is a procedure, just write “procedure”; you don't have to show the form in which Scheme prints procedures. **Also, draw a box** **and pointer diagram for the value produced by each expression.**

(append '(I will return) 'yes '(I will return))

ERROR

(cons '(well I hope you come) '(and see me in the movies))

((well I hope you come) and see me in the movies)

(append (cons 'she '(loves you)) '(yeah (yeah) yeah))

(she loves you yeah (yeah) yeah)

(list 'from 'Liverpool '(to Tokyo) (cons '(what a way to go) '()))

(from Liverpool (to Tokyo) ((what a way to go)))

Take off half a point for every mistake. I was lazy to draw out box and pointers. If you can’t figure it out ask Pedro. Also take off one point if the student forgets to add the starting arrows for every box and pointer.

2. (3 points) Eight TAs are trying to write a midterm. Brian decides that the problems should be represented using his “problem” ADT, which uses the constructor provided below:

(define (make-problem question solution points)

(list (list question solution) points))

(a) Write selectors for this ADT.

(define question caar)

(define solution cadar)

(define points cadr)

(b) The exam has to not be worth too many or too few points. Write a procedure total-points which takes a list of problems as its argument, and returns the sum of their point values.

(define (total-points probs)

(if (null? probs)

0

(+ (points (car probs)) (total-points (cdr probs))))

(c) Brian decides to add an expected-time attribute to problems by using the following new constructor:

(define (make-problem question solution points expected-time)

(list (list question solution) expected-time points))

Assuming the selectors are changed accordingly, what else, if anything, would you need to change to make your answer in part (b) still work?

Nothing

One point each.

3. (4 points) The procedure addup, shown later, takes two arguments: a list of numbers, and a *goal* number. It returns a list of numbers, a subset of the original list, whose sum is the goal number, or #f if there is no such list. It returns the shortest possible list.

> (addup '(2 6 3 4 5) 10)

(4 6)

> (addup '(2 3 4 5) 10)

(5 3 2)

**You do not have to write addup or figure out its algorithm.**

We have written it for you; your job is to modify the program to use data abstraction, as follows:

The program works by maintaining a **queue** of **trials**. Each trial is a list of three elements: a list of the numbers being tried, their sum, and a list of the remaining numbers from the argument list that aren't in this trial. For each trial in the queue, the program checks whether the sum is equal to the goal. If so, that's the solution. If not, the program appends to the queue all the trials that can be made from this trial by adding one more number.

Your job is to improve the readability of this program by designing an abstract data type for **trials**, writing the necessary constructor(s) and selector(s), and rewriting the procedures on the next page to use them correctly.

(a) Write the constructor(s) and selector(s) here:

(define (make-trial tried sum remaining) (list trial sum remaining))

(define tried car

(define sum cadr)

(define remaining caddr)

Here is the code you are modifying:

(define (addup nums goal)

(addup-helper (list (list '() 0 nums)) goal))

(define (addup-helper queue goal)

(cond ((null? queue) #f)

((eq? (cadar queue) goal) (caar queue))

(else (addup-helper

(append (cdr queue)

(map (lambda (new)

(list (cons new (caar queue))

(+ new (cadar queue))

(remove new (caddar queue))))

(caddar queue)))

goal))))

(b) Rewrite addup and addup-helper to use your abstract data type:

(define (addup nums goal)

(addup-helper (list (make-trial '() 0 nums)) goal))

(define (addup-helper queue goal)

(cond ((null? queue) #f)

((eq? (sum (car queue)) goal) (tried (car queue)))

(else (addup-helper

(append (cdr queue)

(map (lambda (new)

(make-trial(cons new (tried (car queue)))

(+ new (sum (car queue)))

(remove new (remaining (car queue)))))

(remaining (car queue))))

goal))))

Half a point off for every mistake in the constructors/selectors and half a point off for every missing correction in addup or extra correction.