Write (above) a pledge that you neither gave nor received help during this exam, and that you adhered strictly to the 60 minute time limit and did not use a computer or any reference material. Sign it.

Abstractions – Exam 1 – 15 January 01

Points per item are indicated in square brackets. You may assume the existence of the following Scheme functions, special atoms and special forms:

car, cdr, cadr, cons, null?, number?, and, or, not, eq?, atom?, equal?, member, list, atom?, list?, map, cond, if, lambda, define, 1+, 1-, zero?, +, -, *, /, <, >, =, (), #t, #f

You may also use as a help function any function which you define on the exam.

1. [10] (first-to-last lat)

Write a function first-to-last which moves the head of a nonempty list to the end. first-to-last rearranges an input list so that the first element of this list becomes the last, the second element (if any) becomes the first, etc. Do whatever seems reasonable if the list is empty.

(first-to-last '(sparkler pinwheel flag)) \Rightarrow (pinwheel flag sparkler)
(first-to-last '(lion)) \Rightarrow (lion)
(first-to-last '(1 2 3 4)) \Rightarrow (2 3 4 1)

2. [10] (7exp n p)

Write a function 7exp which accepts two integers, n and p, and returns the value of seven times n raised to the p power.

(7exp 3 2) \Rightarrow 63
(7exp 10 0) \Rightarrow 7
(7exp 4 3) \Rightarrow 448
3. [10] \((\text{count a lat})\)

Write a function \(\text{count}\) so that \((\text{count a lat})\) returns the number of times the atom \(a\) appears in the lat \(\text{lat}\).

\[
\begin{align*}
(\text{count 'p '(p u p))} & \implies 2 \\
(\text{count 's '(m i s s i s s i p p i))} & \implies 4
\end{align*}
\]

4. [10] \((\text{delete-all bad-atoms lat})\)

Write a function \(\text{delete-all}\) which accepts two \(\text{lat}\)s. The first lat represents a list of “bad” atoms – atoms that we wish to remove from the second lat. \(\text{delete-all}\) should remove all occurrences of any atom in the first lat from the second lat.

\[
\begin{align*}
(\text{delete-all '(a b)' (a b c d e ab c d e a)}) & \implies (c d e ab c d e) \\
(\text{delete-all '(d e)' (a b c d e)}) & \implies (a b c)
\end{align*}
\]

5. [10] \((\text{count-unique-atoms lat})\)

Write a function \(\text{count-unique-atoms}\) that accepts a lat and returns a list of pairs. Each pair consists of an atom in the lat followed by a count of how often it appears. Each individual atom in the lat appears only once in the list of pairs. The order of the returned list does not matter in this question.

\[
\begin{align*}
(\text{count-unique-atoms '(m i s s i s s i p p i))} & \implies ((m 1) (i 4) (s 4) (p 2)) \text{ in some order} \\
(\text{count-unique-atoms '(b i l l c l i n t o n))} & \implies \\
((b 1) (i 2) (l 3) (c 1) (n 2) (t 1) (o 1)) & \implies ((b 1) (i 2) (l 3) (c 1) (n 2) (t 1) (o 1)) \text{ in some order}
\end{align*}
\]
6. [10] Use `fold` to define the procedure `(sum-of-squares l)`, which takes a list of numbers and produces the sum of their squares.

```
(define fold
  (lambda (op base lat)
    (cond
      ((null? lat) base)
      (else (op (car lat) (fold op base (cdr lat)))))))

(sum-of-squares '(1 2 3 4)) ⇒ 30
```

7. [10] (andmap pred l)

Write a function `andmap` which accepts a predicate function of one argument and applies it to each element of an input list `l`. `ormap` should return `#t` if the predicate evaluates to true on all of the elements of `l`.

```
(andmap null? '(a b c e)) ⇒ #f
(andmap number? '(a b 3 (a))) ⇒ #f
(andmap list? '((3 a b) () (a b) (c))) ⇒ #t
(andmap null? '()) ⇒ #t
```
8. Consider the following function definition:

\[
\text{(define list-recur*} \\
\quad \text{(lambda (seed atom-fn list-fn l)} \\
\qquad \text{(cond \{(null? l) seed\} \{\text{atom? (car l)} \{atom-fn (car l) \text{(list-recur* seed atom-fn list-fn (cdr l))}\} \text{else \text{(list-fn (list-recur* seed atom-fn list-fn (car l)) \text{(list-recur* seed atom-fn list-fn (cdr l))})})))}}
\]

\text{list-recur*} can be used to define almost any function that completely traverses a list, though not necessarily as efficiently. For example, you can define \text{islat?} and \text{member*} by

\[
\text{(define islat?} \\
\quad \text{(lambda (lat)} \\
\qquad \text{(list-recur* \#t (lambda (x y) (lambda (x y) \#f) lat))})
\]

\[
\text{(define member*} \\
\quad \text{(lambda (a l)} \\
\qquad \text{(list-recur* \#f (lambda (x y) (or (eq? a x) y)) \text{(lambda (x y) (or x y)) l))})
\]

Define each of the following functions to be the result of a call to \text{list-recur*}: 

(a) [10] \text{(insertl* new a l)} inserts the atom \text{new} in front of all occurrences of atom \text{a} in list \text{l}.

\[
\text{(insertl* \text{'very '}(x (y x z) y x z)) } \Rightarrow \text{(very x (y very x z) y very x z)}
\]

(b) [10] \text{(prod* l)} takes a list of numbers and returns their product:

\[
\text{(prod* \text{'((1 2) (3 (4 5)))}) } \Rightarrow \text{120}
\]

(c) [10] \text{(remove* test l)}, where \text{test} is a boolean function of 1 argument, applies \text{test} to each atomic element of \text{l}, and keeps only those elements for which \text{test} is false.

\[
\text{(remove* (lambda (x) (> x 2)) \text{'((1 2) 1 2 (3 0 2 1))) } \Rightarrow \text{((1) 1 (0 1))}
\]