CS 61A
Interpreters, Tail Calls, Macros, Streams, Iterators
Spring 2019

Guerrilla Section 5: April 20, 2019

1 Interpreters

Questions

1.1 Determine the number of calls to **scheme_eval** and the number of calls to **scheme_apply** for the following expressions.

```
> (+ 1 2)
3
```

```
> (if 1 (+ 2 3) (/ 1 0))
5
```

```
> (or #f (and (+ 1 2) `apple) (- 5 2))
apple
```

```
> (define (add x y) (+ x y))
add
> (add (- 5 3) (or 0 2))
2
```

2 Tail Calls

Questions

2.1 For the following procedures, determine whether or not they are tail recursive. If they are not, write why not and rewrite the function to be tail recursive on the right.

```
; Multiplies x by y
(define (mult x y)
    (if (= 0 y)
        0
        (+ x (mult x (- y 1)))))

; Always evaluates to true
; assume n is positive
(define (true1 n)
    (if (= n 0)
        #t
        (and #t (true1 (- n 1)))))

; Always evaluates to true
; assume n is positive
(define (true2 n)
    (if (= n 0)
        #t
        (or (true2 (- n 1)) #f)))

; Returns true if x is in lst
(define (contains lst x)
    (cond
        ((null? lst) #f)
        ((equal? (car lst) x) #t)
        ((contains (cdr lst) x) #t)
        (else #f)))
```

2.2 Tail recursively implement **sum-satisfied-k** which, given an input list **lst**, a predicate procedure **f** which takes in one argument, and an integer **k**, will return the sum of the first **k** elements that satisfy **f**. If there are not **k** such elements, return 0.

```
; Doctests
scm> (define lst `(1 2 3 4 5 6))
scm> (sum-satisfied-k lst even? 2)  ; 2 + 4
6
scm> (sum-satisfied-k lst (lambda (x) (= 0 (modulo x 3))) 10)
0
scm> (sum-satisfied-k lst (lambda (x) #t) 0)
0

(define (sum-satisfied-k lst f k)




)
```

2.3 Tail-recursively implement **remove-range** which, given one input list **lst**, and two nonnegative integers **i** and **j**, returns a new list containing the elements of **lst** in order, without the elements from index **i** to index **j** inclusive. For example, given the list (0 1 2 3 4), with $\mathbf{i} = 1$ and $\mathbf{j} = 3$, we would return the list (0 4). You may assume $\mathbf{j} > \mathbf{i}$, and **j** is less than the length of the list. (Hint: you may want to use the built-in **append** function, which returns the result of appending the items of all lists in order into a single well-formed list.)

```
; Doctests
scm> (remove-range (0 1 2 3 4) 1 3)
(0 4)

(define (remove-range lst i j)
```

3 Macros

Questions

3.1 What will Scheme display? If you think it errors, write Error

```
> (define-macro (doierror) (/ 1 0))


> (doierror)


> (define x 5)


>(define-macro (evaller y) (list (list 'lambda '(x) 'x) y))


> (evaller 2)
```

3.2 Consider a new special form, **when**, that has the following structure:

```
(when <condition> <expr1> <expr2> <expr3> ... )
```

If the condition is not false (a truthy expression), all the subsequent operands are evaluated in order and the value of the last expression is returned. Otherwise, the entire **when** expression evaluates to **okay**.

```
scm> (when (= 1 0)(/1 0) 'error)
okay
scm> (when (= 1 1) (print 6) (print 1) 'a)
6
1
a
```

Create this new special form using a macro. Recall that putting a dot before the last formal parameter allows you to pass any number of arguments to a procedure, a list of which will be bound to the parameter, similar to (*args) in Python.

```
; implement when without using quasiquotes
(define-macro (when condition . exprs)
    (list 'if ____________________________________________________________________________________)

; implement when using quasiquotes
(define-macro (when condition . exprs)
    `(if ____________________________________________________________________________________)
```

4 Streams

Questions

4.1 What Would Scheme Display?

```
> (define a (cons-stream 4 (cons-stream 6 (cons-stream 8 a))))
> (car a)

> (cdr a)

> (cdr-stream a)

> (define b (cons-stream 10 a))
> (cdr b)

> (cdr-stream b)

> (define c (cons-stream 3 (cons-stream 6)))
> (cdr-stream c)
```

4.2 Write a function **merge** that takes in two sorted infinite streams and returns a new infinite stream containing all the elements from both streams, in sorted order.

```
(define (merge s1 s2)




)
```

5 Iterators

Questions

5.1 What is the definition of an iterable? What is the definition of an iterator? What is the definition of a generator?

5.2 What Would Python Display?

```
>>> def g(n):
        while n > 0:
            if n % 2 == 0:
                yield n
            else:
                print('odd')
            n -= 1
>>> t = g(4)
>>> t

>>> next(t)

>>> n

>>> t = g(next(t) + 5)

>>> next(t)
```

5.3 Write a generator function textbfgen_inf that returns a generator which yields all the numbers in the provided list one by one in an infinite loop.

```
>>> t = gen_inf([3, 4, 5])
>>> next(t)
3
>>> next(t)
4
>>> next(t)
5
>>> next(t)
3
>>> next(t)
4
def gen_inf(lst):
```

5.4 Write a function **nested_gen** which, when given a nested list of iterables (including generators) **lst**, will return a generator that yields all elements nested within **lst** in order. Assume you have already implemented **is_iter**, which takes in one argument and returns **True** if the passed in value is an iterable and **False** if it is not.

```
def nested_gen(lst):
    '''
    >>> a = [1, 2, 3]
        >>> def g(lst):
        >>>     for i in lst:
        >>>             yield i
        >>> b = g([10, 11, 12])
        >>> c = g([b])
        >>> lst = [a, c, [[[2]]]]
        >>> list(nested_gen(lst))
        [1, 2, 3, 10, 11, 12, 2]
    '''
```