cs1120: Exam 1

Due: Wednesday, 06 October at 3:30pm (in class)

UVA ID (e.g., wrw6y) : ANSWER KEY

Directions

Work alone. You may not discuss these problems or anything related to the material covered by this exam with anyone except for the course staff between receiving this exam and turning it in.

Open resources. You may use any books you want, lecture notes, slides, your notes, and problem sets. You may ***not*** use DrScheme or DrRacket, but it is not necessary to do so. You may also use external non-human sources including books and web sites. If you use anything other than the course books, slides, and notes, cite what you used. You may not obtain any help from other humans other than the course staff.

Answer well. Answer all questions 1-9 (question 0 is your UVA ID in two places, which hopefully everyone will receive full credit for), and optionally answer questions 10-11.

You may either: (1) print out this exam and write your answers on it or (2) write your answers directly into the provided Word template and print the result out. Whichever one you choose, you must turn in your answers printed **on paper** and they must be clear enough for us to read and understand. You should not need more space than is provided to write good answers, but if you want more space you may attach extra sheets. If you do, make sure they are clearly marked.

The questions are not necessarily in order of increasing difficulty, so if you get stuck on one question you should continue on to the next question. There is no time limit on this exam, but it should not take a well-prepared student more than a few hours to complete. It may take you longer, though, so please do not delay starting the exam. There is no valid excuse (other than a medical or personal emergency) for running out of time on this exam.

No "snow jobs". If you leave a question **blank**, you will receive **three** points for it. If you have no idea and waste our time with long-winded guessing, we will be less sanguine and the grading will be more sanguine. :-)

Use any Scheme procedure from class. In your answers, you may use any Scheme procedure that appears in the lecture notes or in the book without redefining it (e.g., length, filter, sort, find-best, etc.). If there are multiple similar names (e.g., map vs. list-map), use whichever you like.

Full credit depends on the clarity and elegance of your answer, not just correctness. Your answers should be as short and simple as possible, but not simpler. Your programs will be judged for correctness, clarity and elegance, but you will not lose points for trivial errors (such as missing a closing parenthesis).

UVA ID *again* (e.g., wrw6y) : ANSWER KEY

Your Scores

0	1	2	3	4	5	6	7	8	9	EC	Total
10	10	10	10	10	10	10	10	10	10	2	100

(Your scores are recorded on the second page so that they are not visible to other students when tests are distributed or passed back.)

1. Consider the following grammar:

S ::= N | N and S
N ::= A B C | D E F
A ::= randall | simon
B ::= darius | phillip | ε
C ::= jackson | cowell
D ::= paula
E ::= julie | ε
F ::= abdul

The start symbol is S. The symbol ε denotes the empty string.
(a) Is the language of this grammar infinite or finite?
(b) Give a string of length six (i.e., six words) that is in the language of the grammar.

(a) Infinite. "S -> N and S" is just like "word -> anti word".

(b) randall darius jackson and paula abdul
or
simon cowell and paula julie abdul
or
many other answers ...

Basically, in order to get six words you must use B -> epsilon or E -> epsilon.

2. Consider the following Scheme definition.

```
(define (territory worker-fun lst)
        (if (null? lst) null
                (append (territory worker-fun (cdr lst))
                        (cons (worker-fun (car lst)) null))))
```

Provide a convincing argument that **`territory`** is not equivalent to **`map`**. (Hint: provide inputs on which they behave differently.)

Territory produces the reverse list compared to what map produces.

```
(territory add2 (list 1 5)) ->     (7 3)
(map add2 (list 1 5))       ->     (3 7)
```

[On an obscure note, map in DrScheme can take more than two parameters, and territory cannot, so some students who mentioned that got credit even though it wasn't the knowledge we were hoping you would demonstrate.]

3. Write a Scheme procedure `no-homework` that accepts a list of strings as input. It should return the same list of strings in the same order, but with all instances of "homework" replaced by "none". For example:

```
> (no-homework (list "locke" "ben" "homework" "sawyer" "homework")
("locke" "ben" "none" "sawyer" "none")
> (no-homework (cons ("claire" null))
("claire")
```

```
(define (no-homework lst)
     (map (lambda (elt)
          (if (eq? elt "homework")
               "none"
               elt
          )
     ) lst
```

Note that using "filter" or otherwise removing "homework" is not correct. You had to replace it with "none". You did not have to use map – you could also write it all out.

4. Define a `make-list-remover` procedure that takes one input, an element, and *produces a procedure* as output. The output procedure is a procedure that takes a list of numbers as input, and produces as output that same list but with all entries matching the original element removed. (Hint: besides "define", how do we make procedures in Scheme?)

For example:

```
> (make-list-remover 5)
#<procedure>
> ((make-list-remover 2) (list 1 2 3))
(1 3)
> ((make-list-remover 7) (list 10 -2 13))
(10 -2 13)
```

```
(define (make-list-remover n)
     (lambda (lst)  ;; this is key!
          (filter (lambda (elt)
               (not (eq elt n))) lst)))
```

Since make-list-remover *returns a procedure*, you must start with "lambda". You did not have to use filter, you could write it all out.

```
(define (make-list-remover n)
     (define (helper lst)
          (if (null? lst) lst
               (if (eq? (car lst) n) (helper (cdr lst))
                 (cons (car lst) (helper (cdr lst))))))
     (lambda (x) (helper x)))
```

5. Define a procedure `nested-sum` that takes one input: a list. The list may contain integer elements, but it may also contain other lists (which may themselves contain other lists, hence "nested"). The procedure should return the sum total of the elements anywhere in the list. For example:

```
> (list? (list 1 2 3))
#t
> (nested-sum (list -9 -8 -7 0 4 5 6))
-9
> (nested-sum (list 1 2 (list 0 (list 4 5)) 7 (list -3 0)))
16
> (nested-sum (list ))
0
> (nested-sum (list -1 (list ) -3))
-4
```

(Hint 1: `rewrite-lcommands`. Hint 2: There are three cases. The list may be empty, its first element may itself be a list, or its first element may be an integer. Two of those three cases involve recursive calls.)

```
(define (nested-sum lst)
      (if (null? lst)
            0
            (if (list? (car lst))
                  (+ (nested-sum (car lst)
                        (nested-sum (cdr lst)))
                  (+ (car lst)
                        (nested-sum (cdr lst)))
            )
      )
)

This was very similar to Example 5.9 on Page 91 of the book.
(defined nested-sum lst)
      (fcompose deep-list-flatten list-sum))
```

6. Answer each of the following questions about function growth. Give an argument that explains each answer. For example, to prove that n is in O($n/2$) you might demonstrate the constants $n_0 = 1$ and $c = 2$. If there is any ambiguity, use the definitions from Chapter 7 of the course book.

Is $n+8$ in O(n) ? Why or why not?

Yes. C=2, N0=8

Is n^4 in O(n^3) ? Why or why not?

No. There is no c such that ...

Is n in Ω(4 n - 3) ? Why or why not?

Yes. Pick c = ¼ and n0 = 1. Notably, you can't pick c=1 and n0=1.

Is log n in Ω(n^3) ? Why or why not?

No. There is no c such that ...

Is $5n^2$ in Θ(n^3) ? Why or why not?

No, because n^2 is not in Omega(n^3). Because there is no c such that ...

7. Consider the following four procedures:

```
(define (map f lst)
      (if (null? lst) null
            (cons (f (car lst)) (map f (cdr lst)))))

(define (my-reverse lst)
      (define (reverse-helper x y)
            (if (null? x) y
                  (reverse-helper (cdr x) (cons (car x) y))))
      (reverse-helper lst null))

(define (revmap-alpha f lst)
      (my-reverse (map f lst)))

(define (revmap-beta f lst)
      (if (null? lst) null
            (append     (revmap-beta f (cdr lst))
                        (list (f (car lst))))))
```

Give the running time of `my-reverse` and `revmap-alpha` and `revmap-beta` in Big Theta **Θ** notation. Assume `f` runs in constant time. Which of `revmap-alpha` and `revmap-beta` is faster? Can there be an asymptotically faster `revmap` procedure? (Hint: either demonstrate "yes" by giving the procedure, or argue that no such faster procedure is possible.)

(a) my-reverse is Θ(n) where n is length of list

Note that there are slower implementations of reverse (e.g., it is possible to do it in n^2 time), but this one is carefully crafted to run in linear time.

(b) revmap-alpha is Θ(n+n) = Θ(n)

First you call map. This takes Θ(n) time. Then you pass the result to my-reverse. That also takes Θ(n) time. So Θ(n+n)=Θ(n). Do not multiply them; the two operations are done sequentially. It's *not* the case that we're calling reverse inside every recursive step of map. Instead, we just call it once, at the "outside".

(c) revmap-beta is Θ(n^2)

(append X Y) takes Θ(X) time. The first argument of append is the result of the recursive call to revemap-beta, which in the worse case is the whole answer – a list of size Θ(n). So each recursive call takes Θ(n) time, and there are Θ(n) recursive calls, for Θ(n^2) worse case

(d) The faster procedure is revmap-alpha

(e) No, you need to consider every element at least once. So if there are n elements, you need Omega(n) time.

8. Consider the task of sorting a music playlist for a portable music player or cell phone. We will represent each song as a cons pair of its artist and its title. We wish to sort the playlist by *title first*, and then by artist. Write a procedure `sort-playlist` that accepts one argument, a list of songs, and returns that same list of songs, but sorted as per the description above. Example:

```
> (define song1 (cons "ABBA" "Mamma Mia")
> (define song2 (cons "Taylor Swift" "Love Story")
> (define song3 (cons "Beyonce" "Single Ladies")
> (define song4 (cons "Randy Newman" "Love Story")
> (define song5 (cons "Lonely Island" "I'm On A Boat"))
> (sort-playlist (list song1 song2 song3 song4 song5))
(      ("Lonely Island" . "I'm On A Boat")
       ("Randy Newman" . "Love Story")
       ("Taylor Swift" . "Love Story")
       ("ABBA" . "Mamma Mia")
       ("Beyonce" . "Single Ladies" )      )
> (string<? "ABBA" "Beyonce")
#t
> (eq? "ABBA" "Beyonce")
#f
```

(No points off if you sort Z-A instead of A-Z. All points off if you sort by artist first.)

```
(define (sort-playlist lst)
     (define (compare s1 s2)
          (if (eq? (cdr s1) (cdr s2))
               (string<? (car s1) (car s2))
               (string<? (cdr s1) (cdr s2)) )
     (sort compare lst) )
```

You could also just use < instead of string<? for full credit. Note that we told you about string<? in the example text above; no need to dig through manuals.

You could also write out insert-sort (or select-sort, etc.) yourself instead of just calling sort. Both were full credit.

9. In the not too distant future (next Sunday A.D.), YouTube, Twitter and Facebook have merged together to form YouTwitFace, one social networking site to rule them all and in the darkness bind them. You have been asked to write a friend analysis procedure for this site. Lists of friends are available. Given such a list, you are to find the person with the most friends.

You should write two procedures. The first, `number-of-friends`, takes as input a lists of pairs of strings (representing friends) and a particular string (the person under consideration) returns an integer (the number of friends of that person). The second, `most-friends`, takes as input a list of pairs of strings (representing friends, as before). It should return the name of the person with the most friends. (This can be done in under 10 lines total.) Example:

```
> (define friends (list (cons “rachel” “monica”)
                        (cons “phoebe” “rachel”)
                        (cons “rachel” “joey”)
                        (cons “monica” “joey”)
                        (cons “monica” “chandler”) ))
> (number-of-friends friends “rachel”)
3
> (number-of-friends friends “chandler”)
1
> (most-friends friends)
“rachel”
```

```
(define (number-of-friends lst who)
     (if (null? lst) 0
          (if (or   (eq? who (car (car lst)))
                    (eq? who (cdr (car lst)))
               (+ 1 (number-of-friends (cdr lst) who))
               (number-of-friends (cdr lst) who))
)))
;; OR
(define (number-of-friends lst who)
     (length (filter (lambda (x) (eq? x who)) lst))))

(define (most-friends lst)
     (define (all-friends lst)
          (if (null? lst) null
               (list-append (list (car (car lst))
                                  (cdr (car lst)))
                    (all-friends (cdr lst)))
     (find-best (lambda (person)
          (number-of-friends lst person))
          (all-friends lst)))
;; you could also sort and take the car of the result
```

10. (Extra credit, two points max, no points for leaving this blank.) Give the running time for your implementation of `most-friends` using Θ notation (or a hypothetical implementation if you did not complete #9). Assume that the friend list contains *F* friend pairs.

Typically Theta(F^2). Find best takes Theta(N*CF) time, where N is the length of the list and CF is the running time of the comparison function. For us, the length of the list of all friends is F. The running time of the comparison function, number-of-friends, is also F. Finally, we also have to find all the friends, but since the list of all friends is of length F, that also just takes F time. Se have Theta(F + F*F) = Theta(F^2).

11. (Zero points.) Do you feel your performance on this exam will fairly reflect your understanding of the course material so far? If not, explain why. You will not lose any points for your answer.