YOUR NAME PLEASE:

Computer Science 201
Exam 1
February 17, 2015
7-8:30 pm

Closed book and closed notes. No electronic devices. Show ALL work you want graded on the test itself. There are 6 problems, worth 10 points each.

For problems that do not ask you to justify the answer, an answer alone is sufficient. However, if the answer is wrong and no derivation or supporting reasoning is given, there will be no partial credit.

GOOD LUCK!
1.(a) (5 points)
Write a Racket procedure (count-all item lst) that takes a list lst and returns the number of top-level elements of lst that are equal? to item.

Use only the Racket procedures and special forms: +, define, lambda, null?, empty?, car, first, cdr, rest, cons, append, list, list?, equal?, if, cond, integer constants, and the quoted empty list '().

Do not define any procedures other than count-all.

Examples:
(count-all 4 '4 5 3 4 4 3 0) => 3
(count-all "hi!" '("hello" "greetings") => 0
(count-all 'a '(a l a s (a n d))) => 2
(count-all '(0) '(1 0) 1 (1 (0)) (0))) => 2
1.(b) (4 points)
Draw a tree of recursive calls, with return values, for the application (sums 0 '(2 3 8 4)), where the procedure sums is defined as follows.

(define (sums n lst)
  (cond
    [(null? lst)
     '()]  
    [(null? (rest lst))
     (list (+ n (first lst)))]
    [else
     (cons (+ n (first lst))
           (sums (+ n (first lst))
                 (rest lst)))]))

1.(c) (1 point)
Is the procedure sums in part 1(b) tail recursive?
Why or why not?
2.(a) (5 points)
Consider the Boolean function \( f \) presented by the following truth table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>( f(x,y,z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Write down a Boolean expression to represent \( f(x,y,z) \) using the variables \( x, y, z \).
(Please use dot for AND, plus for OR and prime for NOT.)

Justify the correctness of your expression.

2.(b) (5 points)
Give a Boolean expression (using AND, OR, NOT as in part 2(a)) corresponding to each of the following statements involving the Boolean variables \( P, Q, \) and \( R \).

You DO NOT need to justify the correctness of your answers.

(i) At least two of \( P, Q, R \) are 0.
2.(b) continued.

(ii) Exactly two of P, Q, and R are 1.

(iii) P, Q and R do not all have the same value.

(iv) P is true or the values of Q and R are different.

(v) If we write out P, Q, R in left-to-right order, the binary number represented is one of the numbers 2, 3, 5, or 7.
3.(a) (6 points)
Define a recursive procedure (deep-remove item lst) that takes an item and a list lst and returns a list that is lst with every occurrence of an element equal? to item removed, no matter how deeply nested.

Use only the Racket procedures and special forms: define, lambda, if, cond, equal?, null?, empty?, car, cdr, first, rest, cons, append, list, list? and the quoted empty list '().

Do not define any procedures other than deep-remove.

Examples:
(deep-remove 3 '()) => '()
(deep-remove 3 '(4 5 3 4 4 3 0)) => '(4 5 4 4 0)
(deep-remove "no" '("no" "yes" "maybe" "no")) => '("yes" "maybe")
(deep-remove 3 '((1 2 (3)) ((1 2 3)) 3 2 1)) => '((1 2 ()) ((1 2)) 2 1)
(deep-remove '(3 4 ) '((3 4 ) ((4 3 )) ((5 6 ) (3 4 )))) => '(((4 3 )) ((5 6 )))
(deep-remove '(3 ) '(3 )) => '3
(deep-remove '(3 ) '(((3 )))) => '()
3.(b) (4 points) Draw a tree of recursive calls for the application \((\text{rec } '((1 2) 3 (4 5)))\), where the procedure \(\text{rec}\) is defined as follows. Give the final value of the application; you do not need to give intermediate values.

\[
\text{(define (rec lst)}
\]
\[
\begin{align*}
\text{(cond} & \\
\quad & [\text{\((<= \text{ (length lst) 1)}\)} \\
\quad & \text{\text{lst}\}]} \\
\quad & [\text{\((\text{list? (first lst))}) \\
\quad & \text{(append (rec (first lst)) (rec (rest lst))})\}]
\quad & \text{\[\text{else}} \\
\quad & \text{(append (rec (rest lst)) (list (first lst))))\}]
\end{align*}
\]
4. For each of the following three Turing machines, assume that it is started in a configuration with a nonempty contiguous sequence of 0’s and 1’s on the tape, surrounded by blanks, with the head on the leftmost non-blank symbol, in state q1. (Recall that the blank symbol is denoted by b.)

For each of the three machines do the following.

(i) Show the first four configurations starting with the initial configuration for the input 100 (counting the initial configuration as the first of the four.)

(ii) Specify for every input string of 0’s and 1’s whether the Turing machine will eventually halt, and the final contents of the tape if and when it does halt.

4.(a) (3 points)

(q1, 0, q1, 0, R)
(q1, 1, q1, 1, R)
(q1, b, q2, b, L)
4.(b) (3 points)

(q1, 0, q1, 1, R)
(q1, 1, q1, 0, R)
(q1, b, q2, 1, L)
(q2, 1, q2, 1, L)
(q2, b, q1, b, R)

4.(c) (4 points)

(q1, 0, q1, x, R)
(q1, 1, q1, 1, R)
(q1, b, q2, b, L)
(q1, x, q1, x, R)
(q2, x, q2, x, L)
(q2, 1, q2, x, L)
(q2, b, q1, b, R)
5. (10 points)

5.(a) State the theorem we proved in class about the Halting Problem for Racket programs. (Note: DO NOT prove the theorem, just state it.)

5.(b) Give an example of the rm command in Linux, and briefly describe what it does.

5.(c) Give a Racket expression, using only cons, numbers, and the quoted empty list '(), that evaluates to '((1) (2 3)).
5. continued.

5.(d) What does the following procedure do when called on positive integers \(x\) and \(y\)?

\[
(\text{define } (\text{what } x y) \\
(\text{if } (\geq x y) \\
(\text{list } x y) \\
(\text{what } y (+ x 2))))
\]

5.(e) For the following procedure find the values of \((f 1)\), \((f 2)\), and \((f 3)\). Rewrite the procedure using the special form \text{let} so that there is only ONE recursive call \((f (- n 1))\).

\[
(\text{define } (f n) \\
(\text{if } (= n 1) \\
4 \\
(* (f (- n 1)) \\
(f (- n 1))))))
\]
6. (10 points) Evaluate the following Scheme expressions. (Remember to show work to permit partial credit!)

Example:

(cons 1 '(6 5)) => '(1 6 5)

(a) (cons '(1 2) '(3 4)) =>

(b) (first (rest '((apple book) (cat) (doubt eel))))) =>

(c) (map length '((3 (2 1)) (5 4 6) ((7) 8))) =>

(d) (let ((lst1 '(3 4 5 6 7 8 9)))
    (let ((lst2 (filter (lambda (x) (and (even? x) (< x 8))) lst1)))
     (append lst2 lst2))) =>

(e) (let ((p (lambda (x) (lambda (q) (q (q x))))))
    (r (lambda (y) (* y y))))
    (list (r 2) ((p 3) r))) =>
Additional space for calculations.
<table>
<thead>
<tr>
<th>problem</th>
<th>points</th>
<th>actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>