University of California, Berkeley – College of Engineering
Department of Electrical Engineering and Computer Sciences
Fall 2003
Instructor: Dan Garcia

CS 3 Quiz

Personal Information

<table>
<thead>
<tr>
<th>Last name</th>
<th>First Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student ID Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The name of your TA (please circle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex Andrew Anil Clint Lauren</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name of the person to your Left</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name of the person to your Right</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

All the work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS3 who have not taken it yet. (please sign)

Instructions

We will drop your lowest score for questions 1 through 4. Question 0 (1 point) involves filling in the front of this page and putting your name on every following page.

You have one hour to complete this quiz. The quiz is open book and open notes, no computers.

Partial credit will be given for incomplete / wrong answers, so please write down as much of the solution as you can.

For these questions you only need the functions from the following sections (listed in the back page of the book): Words and Sentences, Arithmetic, True and False and Variables.

Use true instead of #t, and false instead of #f. We have found that handwritten #t and #f unfortunately look too much alike.

Please turn off all pagers, cell phones and beepers. Remove all hats & headphones.

Grading Results

<table>
<thead>
<tr>
<th>Question</th>
<th>Max. Pts</th>
<th>Points Earned</th>
<th>Difficulty (0=easy 5=hard)</th>
<th>Fairness (0=fair 5=unfair)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtotal</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min (of 1-4)</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please comment above & below:
Write the difficulty and fairness ratings above and please add additional comments ← on the left here.
Name:

**Question 1: Shotgun... Set... Hike! (8 points, 1 point each)**

Fill in the blanks below. If something is impossible, write IMPOSSIBLE. If something will produce an error, write ERROR. You do not have to explain the error. The symbol ⇒ means "evaluates to". For example,

\[(+ 3 4) ⇒ \]

Assume the following procedure has already been defined:

\[
\text{(define (maybe x) (if x \text{nd} (se 'yes)))}
\]

1) \((\text{maybe 'false}) ⇒ \) _____________

2) \((\text{first } _ _) ⇒ \text{go-bears!} \)

3) \((\text{first } _ _) ⇒ \text{go-stanford!} \)

4) \((\text{equal? '(a c) '(a "c")}) ⇒ \) _____________

5) \((\text{equal? '(a c) (se 'a '() 'c)}) ⇒ \) _____________

6) \((\text{item 2 'cal)}) ⇒ \) _____________

7) \((\text{bf (bl (bf 'green eggs and ham))}) ⇒ \) _____________

8) \((\text{member? (first 'scheme) '(s c h e m e)}) ⇒ \) _____________

**Question 2: Are Ben and Eva taking a CS3 Quiz? (8 points)**

Write a function called ask-me that combines two sentences to form a question. The first sentence contains two people's first names, like '(clint dan). The second sentence contains an English clause, like '(playing golf). The resulting sentence from ask-me should be formed the following way:

- a) It should begin with the word are.
- b) The two names come next with the word and between them.
- c) The English clause comes last, but the last word in the clause should end with a question mark '?'

Examples:

\[(\text{ask-me '(alex andrew) '(jogging)}) ⇒ (are alex andrew jogging?)]

\[(\text{ask-me '(clint dan) '(playing golf)}) ⇒ (are clint and dan playing golf?) \]

Write ask-me below: (you shouldn't need to use 'if' or 'cond')

\[
\text{(define (ask-me names clause)}
\]
Name: ________________________________

**Question 3:** "You were weaving, sir..." "Weave?! I can't even knit!" (8 points)

We're going to create a new data type called a `traffic-report` to store information about an intersection. A `traffic-report` contains two pieces of information: a word representing the color of the light (green, yellow, or red) and a word describing who is behind us. Here is the constructor:

```
(define (make-traffic-report light behind)  
  (se '(light is) light behind 'is-behind-me))
```

a) Now, it's your job to define the selectors for a `traffic-report`. (3 pts)

```
;; Takes a traffic-report and returns a word representing the color  
;; of the traffic light (green, yellow, or red)  
(define (get-light tr) ;; tr stands for traffic-report

;; Takes a traffic-report and returns a word describing who is behind us  
(define (get-behind tr) ;; tr stands for traffic-report
```

b) The function `driving-advice` takes a `traffic-report` & returns one of three words:
   It returns the word go if the light is green or...
   if the light is yellow and there is not a cop behind us.
   It returns the word careful if the light is yellow and there is a cop behind us.
   It returns stop if the light is red, no matter what. Examples:

```
(driving-advice (make-traffic-report 'green 'cop)) => go
(driving-advice (make-traffic-report 'yellow 'school-bus)) => go
(driving-advice (make-traffic-report 'yellow 'cop)) => careful
(driving-advice (make-traffic-report 'red 'cyclist)) => stop
```

Define `driving-advice`. **You may use one cond but no if's in your solution!** (5 pts)

```
(define (driving-advice tr) ;; tr stands for traffic-report

```

"Numbered lines are used only for tracking purposes."
Question 4: Vegas baby, yeah yeah! (8 points)

Below is a definition of greatest-rank. Given three card ranks, greatest-rank should return the greatest of the three. We have utilized outranks? below. Just to refresh your memory, outranks? takes two cards ranks and returns #t if the first rank is greater than the second and #f otherwise. Assume outranks? works correctly.

```
(define (greatest-rank r1 r2 r3)
  (if (or (outranks? r1 r2) (outranks? r1 r3))
      r1
      (if (outranks? r2 (and r3 r1'))
        r2
        r3)))
```

There are two bugs. Identify both of them by filling in the blanks below.

Without any fixes, (greatest-rank

Changing line

fixes that bug.

Now, assume we make the change you suggest. There is still one remaining bug.

After the change, (greatest-rank

Changing line

fixes that bug so it works correctly for all valid input.

You're done!!