CSU2500 Exam 2 HONORS SUPPLEMENT – Fall 2009

Name: ________________________________

Student Id (last 4 digits): ________________________________

- This supplement to Exam 2 is intended for students enrolled in the Honors section of 2500.

- See the instructions on the regular exam.

*Good luck!*

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>/out of</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/ 5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>/ 5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>/ 15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>/ 15</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>/ 20</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1 Design the function even-frogs? that takes a list of symbols and returns true if the symbol ’frog occurs in the list an even number of times. Define the function using a loop function.
Problem 2 Recall that the contract for a Church numeral is

\[ [X \rightarrow X] \rightarrow [X \rightarrow X] \]

For example, the Church numeral for 3 is

\[
\text{\textit{lambda}} (f) (\text{\textit{lambda}} (x) (f (f (f x))))
\]

1. Write \( \text{cn+1} \), the Church-numeral equivalent of \( \text{add1} \).

2. Write a pair of Scheme functions to convert between Church numerals and regular Scheme non-negative integers: \( \text{cn->int} \) and \( \text{int->cn} \).
Problem 3  An oracle is a function that knows about a number and can respond
to guesses about the number. Here is our data definition for Oracles:

;;; An Answer is one of:
;;; - 'low
;;; - 'high
;;; - 'ok
;;; An Oracle is a [Number -> Answer].

Th oracle fred, for example, knows about the number 5:

(fred 3) ; produces 'low
(fred 4) ; produces 'low
(fred 5) ; produces 'ok
(fred 6) ; produces 'high
(fred 7) ; produces 'high

1. Design a function number->oracle that makes an oracle for a given
number.

2. Design a function oracle->number that consumes an oracle and two
integers lo, hi, and produces the number the oracle knows. Assume that
lo < hi, and that the number known to the oracle is an integer in the range
[lo, hi).

Your function must be efficient; it should only make at most about 20
guesses in order to find a number in the range [0, 1000000).
[Here is some more space for the previous problem.]
Problem 4 Recall that in a previous problem, we defined sets with the data definition

```plaintext
;; A [Setof X] is a [Listof X]
;; No repetitions allowed.
```

and we then added an element-comparison function as an extra argument to our set-processing functions, such as `contains?`.

One reason we need to add an element-comparison argument to set functions is that we can’t otherwise handle sets whose elements are themselves sets. For example, consider the set \{1,3,5\}, represented by the list `'(1 3 5)`. Is it a member of the following set-of-sets-of-numbers?

```
'((2 4 6) (5 3 1) (42)) ; A [Setof [Setof Number]]
```

Yes, it *is* a member of the set—it’s the second item—but it’s represented as `'(5 3 1)`, so `equal?` won’t find it for us.

All of this would straighten out nicely if we just went ahead and defined a function `set=?` for comparing sets that took a third argument for comparing elements of the set.

1. Define this function. (Hint: you might do well to define a helpful auxiliary function.) Don’t write recursive code in your solution; use loop functions.

2. Now define `numsetset=?`, which determines if two sets-of-sets-of-numbers are equal. It only takes two arguments, of course, not three.
[Here is some more space for the previous problem.]