Question 1 (3 points):

What will Scheme print in response to the following expressions? Assume that they are typed in sequence, so definitions affect later interactions. If an expression produces an error message, you may just say "error"; you don't have to provide the exact text of the message. If the value of an expression is a procedure, just say "procedure"; you don't have to show the form in which Scheme prints procedures.

```
(lambda (x y) x)

(* (+ 2 3) (+ 2 6))

(first '(help!))

((word 'but 'first) 'plop)

(let ((+ -)
      (- +))
  (- 8 2))

((lambda (z) (- z z)) (random 10))
```

Question 2 (1 point):

Check off all of the following problems that can be solved in linear time:

(a) Find the largest of a sentence of numbers.

(b) Find out whether a sentence contains two equal words (for example, the sentence `(the cat in the hat)` contains `the` twice).

(c) Find out whether or not *all* the words in a sentence are equal.

Question 3 (5 points):

You're a bureaucrat at the CIA. Someone has filed a Freedom of Information Act request for some of your secret documents. Your plan is to release the information, but with all the numbers blanked out, so that for a sentence such as

```
We have 43 tanks ready to roll at 6 pm tomorrow
```

you'll release this version:

```
We have OMITTED tanks ready to roll at OMITTED pm tomorrow
```

Write a procedure called `cia` that takes a sentence as its argument and returns the censored version as in the example above.

Don't worry about capital versus lower case letters!

Question 4 (5 points):

(a) Write a procedure called `test-both` that takes two predicate functions as its arguments, returning a new predicate function whose result is true if and only if both of the given predicates would return true for the same argument. For example:

```
> (define positive-integer? (test-both integer? (lambda (x) (> x 0))))
POSITIVE-INTEGER?
> (positive-integer? -6)
#F
> (positive-integer? 3)
#T
> (positive-integer? 'ringo)
#F
```

The argument predicates and the returned predicate should all take one argument. **Read the problem again; don't write** `positive-integer?`**!**

(b) Fill in the values returned by the following interactions:

```
> (define foo (test-both > =))

> (foo 8 5)

> ((test-both empty? word?) '())

> ((test-both number? 5) 2)
```

Question 5 (5 points):

Write a procedure `can-add?` that takes two arguments: a number and a sentence of numbers. It should return true if and only if the first argument is the sum of exactly two of the numbers in the second:

```
> (can-add? 7 '(5 3 8 2 6))
#T
> (can-add? 7 '(5 3 8 7 6))
#F
```