Name:

CSE 6341 **Mid-term #2** **Sample.**

(Closed book, closed notes, closed neighbor.)
(A single "cheat sheet" is allowed; if you use one, write your name on it, and turn it in with your exam.)

Keep your answers brief and to the point. Rambling, irrelevant discussions may be penalized. If a question (or part thereof) does not make sense, ask me for clarification; if it still does not make sense, say so, and explain why it does not make sense.

1. We said that *functions* and *forms* (or *special forms*) have to be handled differently by the Lisp interpreter.
   a. Why was this? Explain precisely.
   b. How is this difference reflected in the Lisp interpreter? Explain precisely. Your explanation must be in terms of the definition of the Lisp interpreter in Lisp, not in terms of the details of your implementation (in Java or Python etc.)

2. This problem asks you to modify the behavior of Lisp `COND` construct. Currently, suppose you evaluate the following:

   ```
   (COND (b1 e1) (b2 e2) ... (bn en))
   ```

   And suppose all the `bi`'s evaluate to `NIL`. Then the interpreter will raise an exception and escape to the top level. The modification you are required to make is the following: if all the `bi`'s evaluate to `NIL`, the interpreter should return the value of `e1` as the value of the entire `COND` construct; if, on the other hand, any of the `bi`'s, say, `b3` evaluates to `T`, then the behavior should be the normal one, i.e., return the value of `e3`. It is only in the case that all the `bi`'s evaluate to `NIL` that it should return the value of `e1` as the value of the entire `COND` construct.

   Your answer *must* be at the Lisp level, *not* in terms of your implementation in Python or Java etc. In other words, you must write Lisp functions *in the design notation* to, possibly, replace one or more of the Lisp functions that are currently part of the Lisp `evaluate[]` mechanism. You may, of course, introduce any additional auxiliary functions you need but, again, they must be Lisp functions written in the design notation. If it is not possible to do this, explain why not.

3. Consider the following results:
   a. $\{\textbf{true}\}\ x := 0\ \{\textbf{false}\}$
   b. $\{\textbf{false}\}\ x := 0\ \{\textbf{true}\}$
   c. $\{\textbf{false}\}\ x := 0\ \{\textbf{false}\}$
   d. $\{\textbf{true}\}\ x := 0\ \{\textbf{true}\}$

   For each one, state whether it is valid or not, along with a brief explanation. For each one, state whether it is derivable or not using the axioms and rules we have seen, and if it derivable briefly indicate how you would derive it.

   Repeat the problem replace all the braces (curly brackets) with angle brackets so that the results are total correctness results rather than partial correctness result (but without derivation of the valid results since we have not yet seen axioms/rules for total correctness).

4. Suppose the behavior of the '**read** X;' command is changed as follows: it discards the first element in IN, then assigns the value of *head*(IN) (without removing that element from IN). So, for example, if before the execution of this command, IN was <2, 3>, after the command finishes IN will be <3> and X will be 3 (not 2). Let OUT[k] denote the k^{th} element of OUT. Will the following be valid?

```
a. { #OUT = 0 } read x; read y; write (x+y) { true }
b. < #OUT = 0 | read x; read y; write (x+y) | true >
```

This sample is a bit too small. There will probably be five questions in the actual midterm.