Name:

CSE 6341 **Mid-term #1** **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.)

There are 5 questions, each worth ??/varying points. The questions are not all equally difficult but none should take you more than about 8 to 10 minutes to answer. Please 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. Suppose B is a BNF grammar and AB an attribute grammar built on B (i.e., by adding appropriate attributes, evaluation rules, and conditions to B). Answer the following questions:
   a. If B is ambiguous can AB be unambiguous?
   b. If B is unambiguous, can AB be ambiguous?
   In each case, if your answer is "no", explain why; if your answer is "yes", give a simple example to justify your answer.

2. Consider a simple block-structured language (no procedures or functions), consisting of simple commands such as $\langle if \rangle$, $\langle while \rangle$, sequential composition, assignment, etc., with $\langle block \rangle$ also being a command. A $\langle block \rangle$, as usual, consists of a sequence of (variable) declarations followed by a $\langle stmt\ seq \rangle$. The standard static scope rule for such a language allows us refer, from inside a block, to the variables of the current block, the surrounding block, etc. and these can be checked using the standard approach we have seen. Suppose we want to impose an additional condition that no variable name may be declared more than three times in a program; note that this condition applies even for blocks that are completely separated from each other, i.e., not nested inside each other. How would you express this condition in the attribute grammar? Explain briefly. Or if this cannot be done, explain what the problem is.

3. Consider the translational semantics (in which we defined the *Code* attribute) that we defined. Suppose we want to add a new type of ⟨*command*⟩ to the language, the **break** command. The **break** command can appear only inside a loop. The effect of executing the **break** (at run-time) should be to transfer control to the point immediately following the innermost loop that the **break** appears in. How would you modify the translational semantics to take care of the **break**? Briefly describe the main changes you would have to make.

4. Write a Lisp function `isIn[S1,S2]` that takes two arguments S1 and S2 both of which are S-expressions (either, both, or neither, might be atomic; one might be a "list", the other not; etc.). `isIn[]` should return T if S1 "appears in" S2 and NIL otherwise. What I mean by "appears in" is that if you wrote them both out (using the dot notation), you will see that S1 textually appears in S2 (ignoring whitespace etc.). Use only the part of Lisp we have discussed in class. If such a function cannot be written, explain why not. If you need to define auxiliary functions, feel free to do so.

5. Define a Lisp function `maxList[L]` that takes a non-empty *list* of numbers and returns the largest number in the list; briefly explain your solution (unless it is obvious how it works!). If this cannot be done in Lisp, explain what the problem is.