

Name:

CIS 655

Sample Mid-term #1
(Closed book, closed notes, closed neighbor!)

There are six questions. Answer all questions. Be clear, precise, and to the point. Rambling, irrelevant discussions will be penalized.

1. (15 points). What, if any, is the relation between *data abstraction* and *abstract* parse trees? Explain briefly.
What, if any, is the relation between *data abstraction* and *recursive descent*? Explain briefly.

2. (15 points). Can a given language have more than one (pure) BNF grammar? If yes, give a simple example. If not, explain briefly why not.

3. (20 points). Some languages (C/C++ etc.) have *conditional expressions*. Suppose we want to add a conditional expression to Core, the language you are writing an interpreter for. A conditional expression can be used wherever an $\langle exp \rangle$ can be, for example, in an assignment statement:

```
X := if (Y > 0) then 25 else (U + Z)
```

When this is executed, it will assign 25 to X if Y is greater than 0, else it will assign the value of (U + Z) to X. Write down a suitable BNF production for $\langle cond\ exp \rangle$. Also write down the corresponding *Exec* or *Eval* procedure. You must use the recursive descent approach; you may use a concrete view of the parse-tree or an abstract view.

4. (15 points). Consider the following condition: "A variable can hold at most three different values (one at a time, of course) during the execution of a program." In other words, once a variable is initialized, you can change its value at most twice. Is this a context-free, context-sensitive, or run-time condition? Explain briefly.

5. (15 points). What exactly does it mean for a grammar to be *ambiguous*? If a grammar is ambiguous, does that cause problems for parsing, or for printing, or for execution? Explain briefly.
6. (20 points). We said that we get better *type safety* in our interpreter by using the approach where we have a class corresponding to each non-terminal and each node in the parsetree is represented as an object of the corresponding class, than if we represent the entire parsetree as a monolithic object. What does this mean? How exactly do we get better type safety? Also, is this true for the case where we follow the principles of data-abstraction or the one where we take a concrete view of the parse tree or both? Explain.