CS 423 Midterm Examination Name: _____

1. (30 points) Suppose that, like many famous languages, you had to support nested comments, so that the comment `/* this is /* a nice */ comment */` was treated as one big comment. Further suppose that such nesting was allowed up to 3 levels deep, but no more. So, for example, the following attempt at a comment would be flagged as an error:

```
/* this is /* a nice /* big /* juicy */ detailed */ organic */ comment */
```

Are these limited-nesting comment rules implementable using Flex regular expressions? If so, give Flex rule(s) that implement these rules, and show how errors are caught and reported. If these comment rules are not implementable, explain (and give examples) why not.

2. (30 points) (a) Write a **regular expression** (you may use Flex extended regular expression operators) for the set of strings described by the context free grammar below. You may use {id} to denote usual regular expression for C/C++ variable names for IDENT, namely [a-zA-Z_][a-zA-Z0-9]*. (b) Under what circumstances is it better to use regular expressions, and under what circumstances is it better to use context free grammars?

```
LVarD : Type VarDs ';'  ;
Type : INT | DOUBLE | BOOL | Name ;
Name : IDENT | QualifiedName ;
QualifiedName : Name '.' IDENT ;
VarDs : VarD | VarD ',' VarDs ;
VarD: IDENT | IDENT '[' ']' ;
```

3. (20 points) Explain how the Bison value stack is used: (a) during a shift operation, (b) during a reduce operation.

4. (30 points) (a) Describe how and when internal nodes need to be constructed, in order for a Bison-based parser to end up with a tree that holds all leaves/terminal symbols. (b) Under what circumstances might a new non-terminal node construction site be skipped? (c) Under what circumstances might some of the leaves/terminal symbols not be needed later during compilation?

5. (40 points) Consider the following grammar for Jzero if statements, given in Bison/YACC syntax. Note that it stops at Expr and Block, giving no non-terminal definitions

```
IfThenStmt: IF '(' Expr ')' Block ;
IfThenElseStmt: IF '(' Expr ')' Block ELSE Block ;
IfThenElseIfStmt: IF '(' Expr ')' Block ElseIfSequence
       |  IF '(' Expr ')' Block ElseIfSequence ELSE Block ;

ElseIfSequence: ElseIfStmt | ElseIfSequence ElseIfStmt ;
ElseIfStmt: ELSE IfThenStmt ;
```

If you feed this grammar fragment into Bison/YACC by itself, a) What are the terminal symbols? b) What are the nonterminal symbols? c) Which nonterminals have recursive productions?\

(d) In Bison grammars, why do some recursive grammar rules use epsilon, while others do not?

6. (30 points) Rewrite the grammar from problem #5 with the following changes. Define Block as a non-terminal consisting of a sequence of zero or more statements enclosed in curly brackets. Allow the use of either a Block or a Statement in each location that currently allows only a Block on the righthand side of the grammar. Define a Statement to be either an IfThenStmt, an IfThenElseStmt or an OtherStmt, followed by a semi-colon. You may leave OtherStmt as a terminal symbol.

7. (30 points) Write C or Java code that is error free and produces no warnings, which performs the following tasks: a) declare a variable that can hold a pointer/reference to a `token`, where a `token` has an integer `category`, a string `lexeme`, and an integer `lineno`, b) allocate a `token` instance from the heap and point your variable at it, and c) initialize your memory to all zero bits. You may assume that the definition of `token` (struct or class) with its member/field definitions has already been defined and visible in the source file before your code fragment.

8. (20 points) How do you debug a Bison (or YACC) grammar? What kinds of bugs are likely to occur, and how are they discovered and fixed?

9. (30 points) What are semantic attributes? Briefly define synthesized and inherited attributes, and give an example of how you might use such semantic attributes to keep track of symbol table information during semantic analysis for a language such as C or Java.

10. (30 points) Symbol tables play a prominent role in semantic analysis. Compare what kinds of scopes are present in the full Java language, with that of our subset language Jzero. Suppose you had to support the notion of package scoping in Jzero such that names are referenced within the current package, and names in other packages can be referenced with (possibly a chain of) names separated by periods, as in the example `package.name` syntax. What might you do in your compiler's symbol tables so that the names defined in various packages can be found when needed?