CS 423 Midterm Examination Name: _____

1. (20 points) Explain why the comments in most programming languages are generally discarded by the lexical analyzer, rather than being returned as a token with an integer category like all the other parts of the source code.

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?

```
declaration : type_specifier decl_list ';'  ;
type_specifier : INT | CHAR | DOUBLE ;
decl_list : decl | decl ',' decl_list ;
decl: IDENT | '*' IDENT | IDENT '[' INTCONST ']' ;
```

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 C struct declarations, given in YACC-style syntax.

```
vd : cl t dl ';' | cl t ';' ;
cl : STATIC | DYNAMIC | /* epsilon */ ;
t : INT | STRUCT '{' vl '}' ;
dl : d | d ',' dl ;
d : IDENT | '*' d | d '[' INTCONST ']' ;
vl : vl vd |  /* epsilon */ ;
```

a) What are the terminal symbols? b) What are the nonterminal symbols? c) Which nonterminals have recursive productions? d) Why might some recursive grammar rules use epsilon, while others do not? e) Rewrite this grammar so that it recognizes the same declarations but does not use epsilon production rules, if that is possible. If not, explain why not.

6. (30 points) Write C code that is error free and produces no warnings, which performs the following tasks: a) declare a variable of type pointer to `struct token`, where `struct token` has an integer `category`, a string `lexeme`, and an integer `lineno`, b) allocate some memory from the heap large enough to hold a `struct token` and point your variable at it, and c) initialize your memory to all zero bits. You may assume that the definition of `struct token` with its field definitions has already been defined earlier in the C file before your code fragment.

7. (20 points) In looking at yydebug output, you might notice that it appeared like the same terminal symbol (for example, a semi-colon) was repeated over and over again in the output, even through sections of parsing where no syntax error occurred. Why might the same terminal symbol appear on the input repeatedly through several iterations of a shift-reduce parser?

8. (30 points) What are semantic attributes? Briefly define and give examples of the major kinds of semantic attributes that might be used in semantic analysis for a language such as C.

9. (30 points) Symbol tables play a prominent role in semantic analysis. Compare what kinds of scopes are present in the ANSI C language, with that of our subset language c113c. Suppose you had to support the notion of "struct label scoping" in c113c such that the names that appear after the reserved word `struct` occupy a name space independent of the name space of declared/defined symbols, such that it would be possible to have a `struct foo` declared at the same level as a variable named `foo`. What might you have to do in your compiler's symbol tables so that the names of these struct labels did not conflict with variable names?