**CS 423 Final Examination** **Name _____**

Answer all questions. This is an open book, open note (take-home) exam.

1. (30 points) Suppose you are writing the lexical analyzer for a lexical analyzer generator tool that is compatible with the GNU flex tool. Write a regular expression for flex-style character sets, which are a shorthand for matching any one character out of several alternatives.

2. (30 points) Write a context free grammar for a subset of C language variable declarations. Your grammar should include any and all combinations of integers, pointers, arrays and structs. You must omit other features such as enums, unions, functions and so on.

3. (30 points) Write a C structure or structures capable of representing the types of variables accepted in problem #2 above.

4. (30 points) Consider the following context free grammar in Bison format for a subset of numeric expressions. You may assume all expressions are of type integer.

```
S : E ;
E : E '+' E | E '*' E | '(' E ')' | ident | const ;
```

where the six terminal symbols are + * ( ) ident const. Terminal symbols denote themselves, except ident (which refers to identifiers, also called variable names) and const (which refers to literal constant values).

a) Give an example that shows the grammar is ambiguous.
b) Add semantic rules to the grammar to compute semantic attributes that can tell for all expressions whether the expression's value may vary at runtime, and if not, what the value of the expression is.
c) Can these attributes be computed at parse time, or must they be computed by a tree traversal after parsing? Explain why.

5. (30 points) Using machine independent three-address instructions, general sample intermediate code for the following for-loop. Variable a is an array of ten integers. Variable i is also an integer.

```
for ( i = 0; i < 5; i++)
   a[i] = a[i] * 3 / 2;
```

6. (20 points) Suppose a language defined a token for an open-ended set of operators, matched by some regular expression like [+\-*/%!=&^]+ that accepts lexemes like ++++ or += or -%-. What extra considerations would be needed in order to use such a token in an expression grammar that supports traditional infix binary and prefix unary operators? For example, the grammar should support expressions such as x + y or -z.

7. (30 points) Describe the use of symbol tables as it relates to the implementation of classes with member variables and member functions. Suppose you had to implement Java classes, with member functions invoked with the syntax x.*f*(*params*) where x is the class instance, f is the member function name, and params are the declared parameters for that function. The semantics of calling x.f(…) are that in addition to the declared parameters, x itself is also passed into f() as as extra parameter, and its fields are visible within the function body of f. Allow for the option of declaring members to be private where they can only be accessed inside class functions. What might you do in your symbol tables in order to support such Java class types and their functions?

8. (20 points) Explain what are addresses in the three-address code used by an intermediate code generator. What information do they represent? Since different kinds of addresses in C can all be typecast back and forth into the type `void *`, can three-address code just use void * for its addresses?

9. (20 points) Write a C structure definition for a symbol table entry. Include fields and auxiliary structures that you would need for semantic analysis and intermediate code generation for a language such as C or Java.

10. (30 points) What are inherited attributes? Give an example of an inherited attribute. What sort of function do you write in order to compute an inherited attribute? Give a pseudocode example of computing an inherited attribute.

11. (30 points) Is type information and type checking in Java computed using a synthesized attribute(s), inherited attribute(s), or both? Under what situations is type information for a tree node obtained by some means other than copying from a parent or child (adjacent) tree node?

12. (20 points) Describe the process of intermediate code generation. What is the objective? What are the input and the output of this phase of compilation?

13. (20 points) Describe the memory regions used at run-time in conventional compiled languages. For each region, describe what sort of information it contains and what determines the lifetime of that information during the program execution.

14. (20 points) Why might a compiler generating x86_64 final code allocate space for all parameters in the activation record ("wasting" up to 48 bytes on the stack for each call), even though the first six parameters are passed in registers?