CS 363 Review Questions for test 1

1. What is meant by an "imperative" language? In other words, how would we distinguish it from a declarative language?

2. The input and output of a compiler is just a text file, a linear collection of statements or instructions. So when we parse, why is it customary for an input program to become represented as a tree, rather than a strictly linear data structure?

3. Name some features or idiosyncrasies of Fortran that are generally not adopted by other languages.

4. What is the difference between an interpreted versus a compiled language?

5. What happens during the pre-processing phase of the compiler?

6. Suppose you ask a compiler to compile your program. What kinds of errors are detected during the
    a. Scanning phase?
    b. Parsing phase?
    c. Semantic checking phase?
    d. Other phases?

7. Suppose we have 2 compilers for the same language: one is top-down and the other is bottom-up. Will they derive the same parse trees? Explain.

8. Distinguish between: intermediate code, machine code, assembly code

9. What are dynamic semantics? Can a compiler check your program for errors in dynamic semantics? If not, how would we check for such errors?

10. Consider the token stream "int dog= 4;". How can we define identifier tokens such that the string "dog" is taken to be a single token, rather than the strings "do" (which is a keyword) and "dog=" (which should be 2 tokens) ?

11. Is the set of valid computer programs a finite or infinite set?

12. What does it mean for a grammar to be ambiguous? Does it matter if a programming language's grammar is ambiguous? If there exists a parse tree with two possible derivations, is this a problem for designing a compiler for the language?

13. A conditional expression as we have seen in Java and other common languages consists of two numbers separated by a relational operator such as <, ==, >, etc. For example, x >= 0 is a conditional expression. For simplicity, let's assume "number" is just a token. Also, a conditional expression can consist of more than one such expression, joined together using

&& and ||. Write a grammar for conditional expressions, and draw a parse tree for x < y && y < z.

14. What is the difference between a derivation and a parse?

15. Explain how to implement table-driven parsing without using an explicit table.

16. What program elements are usually defined using a regular expression? Why can't all program elements be defined using regular expressions? Why not use a context-free grammar to define everything?

17. What can cause a grammar to be unsuitable for top-down parsing? For bottom-up parsing?

18. Does declaring variables help the programmer, or is it only a convenience to the compiler?

19. Re-write the following definition of V so that we don't a "common prefix problem" for top-down parsing: `V → id | id , V`
    Also: why might this transformation be necessary?

20. If a symbol is "overloaded," how does the compiler decide the correct version of the symbol to use? Note there are 3 kinds of symbols we can overload: operators, functions and enumerated constants.

21. What are 2 techniques we have seen where a programming language allows a programmer to share variables outside the scope where they are declared, but not necessarily visible to the entire program?

22. What information is typically stored on the run-time stack? Why do we need a run-time stack? And why is it a *stack* rather than a queue?

23. What are some advantages and disadvantages to doing de-allocation of dynamic objects (i.e. garbage collection) manually?

24. Explain how the "buddy system" of dynamic memory allocation works.

25. Why do some programming languages allow for a "forward" declaration?

26. Why does an expression grammar need to take the associativity of an operator into account? Illustrate with an example.

27. For the following grammar (which you have seen in an earlier handout), compute the first( ) of the nonterminals, the follow( ) of all the grammar symbols, and the predict( ) of both productions. Assume that "NUM" is a terminal (token). How is knowing the first, follow and predict information useful in parsing?

```
type → simple | "*" type | "array" "[" simple "]" "of" type
simple → "int" | "char" | NUM .. NUM
```