1. Advantages of compilers over interpreters:
   - Code generated by a compiler generally works faster than code generated by an interpreter
   - Once a code is compiled, it can be reused multiple times

2. (a) In the production rule “term : term * factor”, why is “term” on the left of the “*” operator?
   (b) What would change if the rule had been written as “term : factor * term” instead?
   - It acts as left recursion
   - It would now change to right recursion

3. Does always using one kind of derivation (right or left) avoid ambiguity problems in CFG?
   - No

4. Types of operator associativity:
   - Left-to-right, Right-to-left, and Non-associative or does-not-apply

5. LL(1) -- Left-to-right scan of i/p stream, Leftmost derivation, 1 token of lookahead

6. LR(k) errors:
   - Reduce/reduce conflict when the top of the stack matches with RHS of multiple production rules
   - Shift/reduce conflict when it is not clear whether to shift a token onto the stack or reduce whatever is at the top of the stack

7. What is not possible in an RE that is possible in a CFG?
   - RE is limited in expressiveness and has no ability to recurse
   - RE cannot define strings that are required to have nested parentheses, brackets, etc or matching pairs
   - RE uses DFA that cannot be used to recognize nesting that is too deep because there won’t be enough states in the DFA. But this is possible with a CFG.

8. Disadvantages of hand-coded compilers:
   - Tedious to write
   - DIfficult to extend and scal
   - Error-prone

9. The semantic analysis phase of a compiler converts:
   - parse tree into an intermediate form

10. Precedence and associativity of operators in a programming language are:
   - Syntactic properties

11. (a) Source-to-source compiling with an example language:
   - Using one compiler to generate a relatively high-level intermediate program and using another compiler to get to the final target program
   - C++ was originally implemented this way

12. Terminal vs Non-terminal symbols in CFG?
   - Terminal symbol represents a single element of the language
   - Non-terminal symbol represents a group of terminal symbols defined by production rules

13. What are bison's precedence directives used for?
   - %left to represent left associativity
   - %right to represent right associativity

14. Is a misspelled keyword a lexical error? Why?
   - No, because it can be an identifier in the program. So, it is a syntactic error

15. Listing grammars in a hierarchy (such as the Chomsky) differentiates them in at least four different ways:
   - Parsing complexity, Expressiveness, Resource requirements, Recognizing Automaton
   - In terms of expressivity, the types of grammars are:
     - Regular, CFG, Context-sensitive, and Recursively enumerable

16. Just because an FA is finite, does the language it accepts have to be finite? Why?
   - No, because as long as the input string is of finite length and satisfies the pattern, it is accepted.
   - e.g. S -> aS, S -> (epsilon) is an example of an infinite language that could be accepted by an FA

17. An example when interpretation may still be required for a compiled language:
   - When "on-the-fly" code generation is desired

18. Bison:
    - Bison is a parse generator for an LALR(1) context-free grammar

19. Two ways associativity may be enforced in a CFG?
    - By having the production rules recurse on the left side
    - Introduce additional production rules to keep the operators separate

20. Enforcing precedence in a CFG?
    - By having nested rules that permit the repetition of only certain operators at each precedence level.
    - e.g. add_op is looser than mul_op as it is written higher in CFG definition

21. What kind of conflict occurs in the following grammar:
    A -> B c d
    A -> E c f
    B -> x y
    E -> x y
    - Reduce/reduce conflict occurs
    - Because when processing “c” from the input stream, the top of the stack matches with the first two rules in the definition.
    - This conflict can be avoided either by refactoring & restating the production rules or by increasing the number of lookahead tokens.

22. For lexical analysis, which is easier to use, DFA, or NFA?
    - DFA is easier to use
    - Although DFA is easier to use, it is not easier to generate a DFA. Hence, first, an NFA is generated that is converted to DFA for use in lexical analysis.

23. Lexical Errors:
    - Badly formed literals, Illegal characters

    Syntactic Errors:
    - Misspelled keywords, Improper structure, Bad statement and expression construct

    Semantic Errors:
    - Undeclared identifiers, mismatched function calls, etc