1. What are three reasons why syntax analyzers are based on grammars?

🡪 Provides a clear and concise syntax description

* The parser can be based directly on the BNF.
* Parsers based on BNF are easy to maintain.

1. What is the difference between lexical analysis and syntax analysis?

The lexical analyzer deals with small-scale language constructs, such as names and numeric literals.

The syntax analyzer deals with the large-scale constructs, such as expressions, statements, and program units.

1. Explain the three reasons why lexical analysis is separated from syntax analysis.

*Simplicity* - less complex approaches can be used for lexical analysis; separating them simplifies the parser

*Efficiency* - separation allows optimization of the lexical analyzer

*Portability* - parts of the lexical analyzer may not be portable, but the parser always is portable

1. What are the primary tasks of a lexical analyzer?

A lexical analyzer is a pattern matcher for character strings

A lexical analyzer is a “front-end” for the parser

Identifies substrings of the source program that belong together - *lexemes*

1. Define *lexeme* and *token*.
   1. Lexeme is the lowest level syntactic unit of a language (eg. \*, sum) and token is the category of lexemes. (eg. Identifier)
   2. Lexemes match a character pattern, which is associated with a lexical category called a *token*
2. Describe briefly the three approaches to building a lexical analyzer.
   1. Write a formal description of the tokens and use a software tool that constructs a table-driven lexical analyzer from such a description
   2. Design a state diagram that describes the tokens and write a program that implements the state diagram
   3. Design a state diagram that describes the tokens and hand-construct a table-driven implementation of the state diagram
3. Discuss state diagrams, finite automata, and lexical analysis.

* State diagrams of the form used for lexical analyzers are representations of a class of mathematical machines called **finite automata**.
* Finite automata can be designed to recognize members of a class of languages called **regular languages**.
* The tokens of a programming language are a regular language, and a lexical analyzer is a finite automaton.

1. Why are character classes used, rather than individual characters, for the letter and digit transitions of a state diagram for a lexical analyzer?
   1. When recognizing an identifier, all uppercase and lowercase letters are equivalent
      1. Use a character class that includes all letters
   2. When recognizing an integer literal, all digits are equivalent - use a digit class
2. How is lexical analysis related to the symbol table?

A lexical analyzer often is responsible for the initial construction of the symbol table, which acts as a database of names for the compiler.

1. What are the two distinct goals of syntax analysis?
   1. Find all syntax errors; for each, produce an appropriate diagnostic message and recover quickly
   2. Produce the parse tree, or at least a trace of the parse tree, for the program
2. Describe the differences between top-down and bottom-up parsers.
   1. *Top down* - produce the parse tree, beginning at the root
      1. Order is that of a leftmost derivation
      2. Traces or builds the parse tree in preorder
   2. *Bottom up* - produce the parse tree, beginning at the leaves
      1. Order is that of the reverse of a rightmost derivation
3. Describe the parsing problem for a top-down parser.
   1. Given a sentential form, xAα , the parser must choose the correct A-rule to get the next sentential form in the leftmost derivation, using only the first token produced by A
4. Describe the parsing problem for a bottom-up parser.

Given a right sentential form, α, determine what substring of α is the right-hand side of the rule in the grammar that must be reduced to produce the previous sentential form in the right derivation

1. Explain why compilers use parsing algorithms that work on only a subset of all grammars.
   1. Compilers use parsers that only work for a subset of all unambiguous grammars, but do it in linear time ( O(n), where n is the length of the input )
2. Describe how a recursive-descent parsing subprogram is written for a rule with a single RHS.

For each terminal symbol in the RHS, compare it with the next input token; if they match, continue, else there is an error

For each nonterminal symbol in the RHS, call its associated parsing subprogram