1. What is Syntax Analysis in a compiler?

Syntax Analysis is the second phase of a compiler where the token stream from the lexical analyzer is analyzed according to the grammatical structure of the programming language to form a parse tree or syntax tree.

2. What is the input and output of the syntax analyze
Input: Tokens from the lexical analyzer
Output: Parse tree or syntax tree

3. What is a context-free grammar (CFG)?

A CFG is a set of recursive rewriting rules used to generate patterns of strings. It defines the syntax rules of a language using terminals, non-terminals, a start symbol, and production rules.

4. What are the two main types of parsing techniques?

- Top-down: Builds parse tree from start symbol to input.
- Bottom-up: Builds parse tree from input back to start symbol.

5. What is top-down parsing?

Top-down parsing starts from the start symbol and tries to derive the input string by applying productions, building the parse tree from top to bottom.

6. Name one common top-down parsing method.

Recursive-descent parsing or Predictive parsing (LL parsing)

7. What are the limitations of top-down parsing?

- Cannot handle left-recursive grammars
- Needs backtracking unless predictive parsing is used
- May require grammar to be in LL(1) form

8. What is bottom-up parsing?
Bottom-up parsing starts from the input and attempts to reconstruct the parse tree by reducing strings to the start symbol, building from leaves to root.

10. What is LR parsing?

LR parsing is a powerful bottom-up parsing technique that reads input Left-to-right and produces a Rightmost derivation in reverse.

11. What does LR stand for in LR parsing?

L – Left-to-right scanning of the input
R – Rightmost derivation in reverse

12. What is SLR parsing?

SLR (Simple LR) parsing is a type of LR parsing that uses follow sets to decide parsing actions. It's easier to implement but less powerful than canonical LR.

13. Why is SLR parsing limited?

Because it uses simple follow sets, it may incorrectly resolve conflicts for certain grammars that canonical LR or LALR can handle.

14. What are the "most powerful" LR parsers?

- Canonical LR (LR(1)) – the most powerful and general
- LALR(1) – Lookahead LR, a practical compromise between SLR and Canonical LR

15. Why are LR parsers preferred in practice?

Because they can handle a larger class of grammars (including left-recursive and ambiguous grammars) and support bottom-up parsing with fewer errors.

16. What is the role of a parsing table in LR parsing?

Parsing tables direct the parser’s actions (shift, reduce, accept, or error) based on the current state and next input token.

17. What is a shift-reduce conflict?
A situation where the parser cannot decide whether to shift the next input symbol or reduce the current symbols on the stack.

18. What is a reduce-reduce conflict?
A situation where the parser has more than one possible reduction at the same point in parsing, leading to ambiguity.

19. Can LR parsers handle ambiguous grammars?
Yes, with careful conflict resolution or precedence rules, LR parsers can handle some ambiguous grammars like those used in expression evaluation.

20. Why are ambiguous grammars sometimes used?

They allow more natural and concise representation of constructs like expressions. Disambiguation is handled by adding precedence and associativity rules.

LALR combines LR parsing power with smaller tables, used in most compiler tools like Yacc.

What are the actions in an LR parsing table?
Shift, Reduce, Accept, Error

A viable prefix is a prefix of a right-sentential form that does not extend beyond the handle of that form. In practical terms, it is any prefix of input that can appear on the parsing stack of a shift-reduce parser without causing a conflict. It ensures that the parser is in a valid state before a reduction.

Parser Type	Shift-Reduce Conflicts	Reduce-Reduce Conflicts	Remarks
LR(0)	Not handled	Not handled	No lookahead; only suitable for very simple, unambiguous grammars
SLR(1)	Partially handled	Not handled	Uses FOLLOW sets; may misresolve conflicts due to limited context
LALR(1)	Mostly handled	Partially handled	Combines similar LR(0) cores; may still conflict if lookaheads overlap
Canonical LR(1)	Fully handled	Fully handled	Most powerful; uses full lookahead; large parsing tables