VIVA QUESTION – ANSWER COMPILER DESIGN

1) What is a compiler?

A compiler is a software program that translates high-level source code into low-level machine code.

2) Name a few phases of a compiler.

Lexical analysis, syntax analysis, semantic analysis, and code generation.

3) What is the main purpose of lexical analysis?

Lexical analysis breaks the source code into tokens for further processing.

4) Explain the difference between a compiler and an interpreter.

A compiler translates the entire source code into machine code before execution, while an interpreter translates and executes the code line by line.

5) Define a lexical analyzer.

A lexical analyzer (or lexer) is a component of a compiler that scans the source code, breaking it into tokens.

6) What is tokenization?

Tokenization is the process of dividing the source code into meaningful units called tokens.

7) Name some common tokens in a programming language.

Keywords, identifiers, operators, and constants are common tokens.

8) What is a symbol table in the context of lexical analysis?

A symbol table is a data structure that stores information about identifiers found in the source code

9) What is the role of syntax analysis in a compiler?

Syntax analysis checks the structure of the source code for syntactic correctness.

10) Define a context-free grammar.

A context-free grammar is a formalism for describing the syntax of a programming language.

11) What is a parse tree?

A parse tree is a hierarchical representation of the syntactic structure of the source code.

12) What is the purpose of the LL (1) parsing table?

The LL (1) parsing table is used to guide a parser in making the right choices during parsing, ensuring unambiguous syntax analysis.

13) Explain the concept of a syntax tree.

A syntax tree is a graphical representation of the syntactic structure of the source code, showing how statements are composed.

14) What is a derivation in the context of context-free grammars?

A derivation is a sequence of production rule applications that generates a string in the language of the grammar.

15) Define ambiguity in the context of parsing.

Ambiguity occurs when a grammar allows multiple parse trees for the same input string.

16) What is the purpose of the FIRST set in syntax analysis?

The FIRST set contains the terminal symbols that can begin a valid derivation of a non-terminal in a grammar.

17) What is the primary goal of bottom-up parsing?

Bottom-up parsing aims to construct a parse tree from the leaves (tokens) to the root.

18) Explain the shift-reduce conflict in parsing.

A shift-reduce conflict occurs in a parser when it can't decide whether to shift (read more input) or reduce (apply a production) during parsing.

19) What is an LR (1) parser?

An LR (1) parser is a type of bottom-up parser that uses one-symbol lookahead to make parsing decisions.

20) Define handle in the context of bottom-up parsing.

A handle is a substring of the input that matches the right-hand side of a production during reduction.

21) Differentiate between top-down and bottom-up parsing.

Top-down parsing starts with the start symbol and works down to the input, while bottom-up parsing starts with the input and builds up to the start symbol.

22) What is a reduce action in a parsing table?

A reduce action indicates that the parser should replace a portion of the input with a non-terminal, applying a production.

23) Explain the concept of a handle in LR parsing.

A handle is a substring of the input that matches the right-hand side of a production during a reduction step in LR parsing.

24) What is the purpose of a lookahead symbol in LR parsing?

A lookahead symbol is used to distinguish between different productions of a non-terminal in LR parsing.

25) What is syntax-directed translation?

Syntax-directed translation is a technique for attaching attributes and actions to the grammar symbols to guide code generation.

26) Explain inherited and synthesized attributes.

Inherited attributes are passed up the parse tree, while synthesized attributes are computed bottom-up.

27) How is a syntax-directed translation scheme related to a parse tree?

A syntax-directed translation scheme associates attributes and actions with the nodes of a parse tree to generate code.

28) What is semantic analysis in the context of syntax-directed translation?

Semantic analysis checks the meaning and consistency of a program's structure using attributes.

29) How does a syntax-directed translation scheme differ from a syntax tree?

A syntax-directed translation scheme associates attributes and actions with the grammar symbols, whereas a syntax tree only represents the syntax of the code.

30) What are the phases of semantic analysis in syntax-directed translation?

Semantic analysis includes type checking, scope resolution, and attribute evaluation.

31) Define type checking in semantic analysis.

Type checking ensures that expressions and variables have compatible data types.

32) What is an abstract syntax tree (AST)?

An AST is a tree-like data structure that represents the hierarchical structure of a program, with nodes corresponding to language constructs.

33) Why is intermediate code generation essential in compilation?

Intermediate code simplifies the process of translating high-level code into machine code and allows for optimization.

34) What is three-address code?

Three-address code represents an operation with at most three operands, often used in intermediate code generation.

35) Name some common intermediate representations.

Three-address code, abstract syntax trees, and quadruples.

36) What is the primary purpose of optimizing intermediate code?

Optimizing intermediate code improves the efficiency of the generated machine code.

37) Why is an intermediate representation (IR) used in compilers?

An IR serves as an intermediate step in compilation that simplifies the translation from high-level source code to machine code and enables optimization.

38) What is a control-flow graph (CFG) in the context of IR?

A CFG is a graphical representation of a program's control flow, showing how statements and branches are connected.

39) Why is it important to study compiler design?

Compiler design is crucial for understanding how programming languages work, and it equips developers with the knowledge needed to build efficient and reliable software.

40) What are the key benefits of using a compiler?

Compilers can improve program execution speed, offer better error checking, and make software more portable by translating high-level code into machine code.

41) Can you explain the relationship between a compiler and an assembler?

An assembler translates assembly language into machine code, while a compiler translates high-level language into machine code, making it more human-readable and maintainable.

42) What is the difference between front-end and back-end phases in a compiler?

The front-end handles tasks like lexical analysis, parsing, and semantic analysis, while the back-end focuses on code optimization and generation.

43) How can you create a simple lexical analyzer?

You can create a simple lexical analyzer using regular expressions and a finite automaton to recognize and classify tokens in the source code.

44) What are the main components of a parser in a compiler?

A parser typically includes a tokenizer (lexer), a parser generator, and grammar rules that define the syntax of a programming language.

45) Explain the process of code optimization in a compiler.

Code optimization involves analyzing the intermediate code to improve its efficiency, reduce execution time, and minimize resource usage.

46) How can you generate machine code from high-level code in the code generation phase?

The code generation phase involves translating the intermediate code into machine code, using various techniques like register allocation and instruction selection.

47) What role does the symbol table play in a compiler?

The symbol table is a data structure used to store information about identifiers, such as variable names, their data types, and their memory locations.

48) How can you handle error recovery in a compiler?

Error recovery techniques include panic mode recovery, synchronization, and reporting meaningful error messages to help developers correct their code.

49) Can you demonstrate the process of lexical analysis using a simple code snippet?

Lexical analysis involves breaking down source code into tokens. For example, in C, "int x = 5;" is broken down into tokens: "int," "x," "=", and "5."

50) Show how a top-down parser processes a given statement in a context-free grammar.

A top-down parser starts with the start symbol and aims to derive the input string, following production rules from left to right.

51) Explain how to construct a FIRST set for a non-terminal in a grammar.

The FIRST set for a non-terminal consists of terminal symbols that can begin the strings derivable from that non-terminal. It is determined by examining the rules for that non-terminal and the FIRST sets of other non-terminals it derives.

52) Can you create a simple LL (1) parsing table for a grammar with a few non-terminals and terminals?

An LL (1) parsing table lists the actions for each combination of non-terminals and terminals, indicating whether to predict, shift, or reduce.

53) Walk us through the process of constructing a LR (1) parsing table for a given grammar.

Constructing an LR (1) parsing table involves creating states and items, determining transitions, and resolving conflicts.

54) Demonstrate the concept of a handle in a given LR parsing example.

A handle is a substring of the input that matches the right-hand side of a production during a reduction step in LR parsing.

55) Show how semantic analysis can be integrated into a syntax-directed translation scheme.

Semantic analysis can include type checking and attribute evaluation, where attributes associated with grammar symbols help in generating code.

56) Provide an example of a syntax-directed translation scheme for a simple programming construct.

For instance, you can demonstrate a syntax-directed translation scheme for assignment statements or arithmetic expressions.

57) Explain the purpose and benefits of using an intermediate representation (IR) in a compiler.

An IR simplifies the translation process and serves as a bridge between high-level code and machine code, making optimization easier.

58) Can you demonstrate the code optimization process with a specific code snippet?

Code optimization involves identifying and applying various techniques like constant folding or common subexpression elimination to improve code efficiency.

59) Show how register allocation is done in the code generation phase.

Register allocation assigns variables to processor registers to optimize execution speed and resource usage.

60) Discuss the steps involved in creating a simple compiler for a basic programming language.

The steps include lexical analysis, parsing, semantic analysis, code generation, and optimization.

61) Question: What is a "reduce-reduce conflict" in parsing?

Answer: A reduce-reduce conflict happens when the parser can apply two or more production rules in the same state, leading to ambiguity.

62) Question: How can you differentiate "syntactic sugar" from regular language constructs in a compiler?

Answer: Syntactic sugar represents language constructs that improve readability without altering the language's expressiveness.

63) Question: What's the role of a "lookahead set" in LR parsing?

Answer: The lookahead set contains terminal symbols that guide parsing decisions in LR parsers, helping to resolve shift-reduce and reduce-reduce conflicts.

64) Question: Define a "nullable" symbol in the context of parsing.

Answer: A nullable symbol can derive an empty string in the grammar, meaning it can be absent from a valid derivation.

65) Question: Explain the concept of "register pressure" in code generation.

Answer: Register pressure is the number of variables that need to be stored in registers during code generation, impacting performance and resource utilization.

66) Question: What's the significance of a "semantic stack" in syntax-directed translation?

Answer: The semantic stack stores intermediate values and attributes during parsing, facilitating semantic analysis and code generation.

67) Question: How is "copy propagation" used in code optimization?

Answer: Copy propagation replaces unnecessary variable copies with references to the original variable, reducing memory and improving performance.

68) Question: What is a "greedy method" in syntax analysis?

Answer: A greedy method selects the longest possible substring that matches a production during parsing, often used in top-down parsers.

69) Question: How does "inlining" impact code optimization?

Answer: Inlining replaces function calls with the actual function code, reducing the overhead of function call and return.

70) Question: What is the key purpose of a "canonical LR (1) item" in LR parsing?

Answer: A canonical LR (1) item represents a state in the parsing table and includes a production rule with a dot indicating the position of the parser in that rule.

THANK YOU!