compiler construction mcqs with answers

Compiler construction MCQs with answers serve as an essential tool for students and professionals in the field of computer science and software engineering. They not only help in assessing one’s understanding of compiler design concepts but also prepare candidates for interviews and competitive exams. This article presents a comprehensive overview of important multiple-choice questions (MCQs) related to compiler construction, along with their answers and explanations.

Understanding Compiler Construction

Compiler construction involves the process of designing and building a compiler, which translates high-level programming languages into machine code. The construction of a compiler is usually divided into several phases:

1. Lexical Analysis - The process of breaking the source code into tokens.
2. Syntax Analysis - The validation of the syntax of the token stream against the grammar.
3. Semantic Analysis - Ensuring that the syntax tree adheres to the semantic rules of the language.
4. Optimization - Improving the intermediate code for efficiency.
5. Code Generation - Converting the optimized intermediate code into machine code.
6. Code Optimization - Further enhancing the machine code for performance.

Multiple Choice Questions (MCQs)

Below are some MCQs related to compiler construction, covering various aspects of the topic.

1. Lexical Analysis

Q1: What is the primary function of a lexical analyzer?
A) To parse the source code
B) To generate intermediate code
C) To convert the source code into tokens
D) To optimize the code

Answer: C
Explanation: The primary function of a lexical analyzer is to read the source

code and convert it into tokens, which are meaningful sequences of characters.

Q2: Which of the following is NOT a type of token?
A) Keyword
B) Identifier
C) Operator
D) Function

Answer: D
Explanation: While keywords, identifiers, and operators are types of tokens, "function" is not a token type but rather a concept that can be represented using tokens.

2. Syntax Analysis

Q3: What does a syntax analyzer do?
A) Checks for syntax errors
B) Converts tokens into machine code
C) Generates intermediate code
D) None of the above

Answer: A
Explanation: The syntax analyzer (or parser) checks for syntax errors by validating the sequence of tokens against the grammar of the programming language.

Q4: Which of the following parsing techniques uses a stack?
A) Top-down parsing
B) Bottom-up parsing
C) Both A and B
D) None of the above

Answer: C
Explanation: Both top-down and bottom-up parsing techniques can utilize a stack to manage the parsing process.

3. Semantic Analysis

Q5: What is the primary goal of semantic analysis in a compiler?
A) To check the correctness of syntax
B) To ensure that the program adheres to the rules of the language
C) To optimize the code
D) To generate machine code

Answer: B
Explanation: The main goal of semantic analysis is to ensure that the program

adheres to the semantic rules of the programming language, such as type checking.

Q6: Which of the following is an example of semantic error?
A) Missing semicolon
B) Using an undeclared variable
C) Mismatched parentheses
D) None of the above

Answer: B
Explanation: Using an undeclared variable is a semantic error because it violates the rules of the programming language, even though the syntax may be correct.

4. Intermediate Code Generation

Q7: What is intermediate code?
A) A high-level programming language
B) A low-level machine code
C) An abstraction between source code and machine code
D) None of the above

Answer: C
Explanation: Intermediate code serves as an abstraction between the source code and machine code, making it easier to optimize and generate final code.

Q8: Which of the following is NOT an advantage of using intermediate code?
A) Easier optimization
B) Portability across different architectures
C) Reduced complexity in code generation
D) Increased execution speed

Answer: D
Explanation: While intermediate code can aid in optimization and portability, it does not inherently increase execution speed; rather, it serves as a step toward generating optimized machine code.

5. Code Optimization

Q9: What is the purpose of code optimization?
A) To make the code more readable
B) To reduce the execution time and/or memory usage
C) To simplify the syntax
D) To generate more code

Answer: B
Explanation: The primary purpose of code optimization is to reduce execution

time and/or memory usage, making the program more efficient.

Q10: Which of the following is an optimization technique?
A) Loop unrolling
B) Code inlining
C) Dead code elimination
D) All of the above

Answer: D
Explanation: All listed options are valid optimization techniques used to improve the efficiency of the code generated by the compiler.

6. Code Generation

Q11: What is the final phase of compiler construction?
A) Lexical analysis
B) Syntax analysis
C) Code generation
D) Semantic analysis

Answer: C
Explanation: Code generation is the final phase of compiler construction, where the optimized intermediate code is translated into machine code.

Q12: The target code produced by a compiler is typically in which form?
A) Assembly language
B) High-level programming language
C) Source code
D) All of the above

Answer: A
Explanation: The target code produced by a compiler is typically in assembly language, which can then be further assembled into machine code.

Conclusion

Compiler construction is a complex yet fascinating area of study in computer science. Understanding the various phases of a compiler, from lexical analysis to code generation, is crucial for developing efficient software. The multiple-choice questions provided in this article serve as a valuable resource for students and professionals looking to test their knowledge in this field.

By regularly engaging with MCQs on compiler construction, individuals can solidify their understanding, identify areas for improvement, and stay prepared for technical interviews and exams. Whether you are a student, a software engineer, or a computer science enthusiast, mastering the concepts

of compiler construction will undoubtedly enhance your programming skills and knowledge.

Frequently Asked Questions

What is the primary purpose of a compiler?

To translate high-level programming language code into machine code or intermediate code.

Which phase of a compiler is responsible for syntax analysis?

The parser phase.

What does LL(1) stand for in the context of parsing?

Left-to-right scanning of the input, Leftmost derivation, with 1 lookahead token.

In compiler construction, what is a 'token'?

A token is a sequence of characters that represents a single logical entity in the source code.

What is the role of the lexical analyzer in a compiler?

To read the source code and convert it into tokens for further processing by the parser.

Which data structure is commonly used to represent the syntax tree in a compiler?

Abstract Syntax Tree (AST).

What is the purpose of semantic analysis in a compiler?

To ensure that the source code adheres to the semantic rules of the programming language, such as type checking.

What is a 'symbol table' in the context of compiler

design?

A data structure used to store information about variables, functions, and objects in the source code.

Compiler Construction Mcqs With Answers