Short Questions

1. What distinguishes a deterministic PDA from a nondeterministic one?
2. How are context-free languages related to PDAs in terms of language recognition?
3. Can every context-free grammar be converted into an equivalent PDA? Explain.
4. What are the implications of the pumping lemma for context-free languages on PDAs?
5. How is the acceptance condition of a PDA defined?
6. Describe the concept of a Universal Turing Machine.
7. How is the Church-Turing thesis relevant to Turing Machines?
8. Can a Turing Machine simulate any other Turing Machine? Explain.
9. Discuss the concept of a Turing Machine with multiple tapes.
10. What is the significance of the tape being infinite in a Turing Machine?
11. What does it mean for a problem to be recursively enumerable?
12. How does Rice's Theorem relate to undecidability?
13. Explain the concept of reduction in the context of undecidable problems.
14. Provide an example of a problem that is recursively enumerable but not decidable.
15. What is the impact of undecidability on the field of computer science?
16. How do PDAs contribute to our understanding of formal languages?
17. In what ways do Turing Machines extend the capabilities of finite automata and PDAs?
18. Describe a real-world application that demonstrates the principles of a Turing Machine.
19. How does the concept of undecidability challenge the boundaries of what computers can solve?
20. Can the concept of nondeterminism in PDAs be applied to practical computing problems?
21. How do non-deterministic Turing Machines compare to deterministic ones in terms of power?
22. Discuss the role of PDAs in the parsing phase of compilers.
23. What are the practical implications of Turing's Halting Problem in modern computing?
24. How might the study of undecidable problems influence future research in algorithms?
25. Explain the role of stack memory in the operation of a Push Down Automaton.

Unit - IV

26. What are the main components of a compiler?

27. Describe the role of the front-end in a compiler.
28. What is the purpose of the back-end in compiler design?
29. How do optimization phases improve a compiler's output?
30. Explain the significance of intermediate representation in compilers.
31. What is the role of a lexical analyzer in a compiler?
32. How does input buffering enhance lexical analysis?
33. Describe the process of token recognition in lexical analysis.
34. What is a lexical analyzer generator, and how is Lex used as one?
35. Explain the significance of regular expressions in lexical analysis.
36. Discuss the challenges faced during the lexical analysis phase.
37. How does a lexical analyzer interact with the syntax analyzer?
38. What are the common errors detected by the lexical analyzer?
39. Explain the concept of token, pattern, and lexeme.
40. How does input buffering affect the efficiency of a lexical analyzer?
41. Introduce the concept of syntax analysis in compiler design.
42. Explain the role of context-free grammars in syntax analysis.
43. What is the process of writing a grammar for syntax analysis?
44. Describe the differences between top-down parsing and bottom-up parsing.
45. What is the significance of LR parsing in syntax analysis?
46. Explain the concept of simple LR (SLR) parsing.
47. Discuss the advancements in LR parsing beyond simple LR.
48. How do parsers handle ambiguous grammars?
49. What are the challenges in implementing a syntax analyzer?
50. How does syntax analysis contribute to the overall process of compilation?
51. What are the key differences between LL and LR parsers?
52. How is a parse tree used in syntax analysis?
53. Explain the concept of recursive descent parsing.
54. What is the role of backtracking in top-down parsing?
55. How does bottom-up parsing differ from top-down parsing in terms of efficiency?
56. Describe the process of handling syntax errors in parsing.
57. What are the implications of left recursion in grammar for parsers?
58. How is ambiguity resolved in LR parsers?
59. What makes LR parsers more powerful than their predecessors?
60. Explain the concept of shift-reduce parsing.
61. How do predictive parsers eliminate the need for backtracking?
62. What is the significance of lookahead tokens in LR parsing?
63. Discuss the role of the parse stack in LR parsing.
64. How can parser generators like Yacc/Bison be used in creating parsers?
65. Explain the differences between LALR parsers and canonical LR parsers.
66. What are the common errors detected during the syntax analysis phase?
67. How does error recovery work in syntax analysis?
68. Describe the concept of abstract syntax trees (ASTs) in compiler design.

69. What are the benefits of using parser generators in compiler construction?
70. How do semantic actions integrate with syntax analysis?
71. Discuss the impact of parsing techniques on compiler optimization.
72. How does the choice of parsing strategy affect compiler performance?
73. What are the considerations for selecting a parser for a new programming language?
74. Explain the role of syntax-directed translation in compiler design.
75. How are parsing techniques applied in other areas of computer science beyond compilers?

Unit - V

76. What is syntax-directed translation in compiler design?
77. Define syntax-directed definitions (SDDs).
78. Explain the different evaluation orders for SDDs.
79. Describe a syntax-directed translation scheme.
80. How are L-attributed SDDs implemented in compilers?
81. Discuss the role of attribute grammars in syntax-directed translation.
82. What are the challenges in implementing syntax-directed translators?
83. How does syntax-directed translation affect code generation?
84. Explain the difference between inherited and synthesized attributes.
85. Provide an example of a syntax-directed translation scheme in action.
86. What is the purpose of intermediate-code generation in a compiler?
87. Describe the structure and variants of syntax trees used in intermediate-code generation.
88. Explain the concept of three-address code in compiler design.
89. How does intermediate-code generation facilitate optimization?
90. Discuss the translation of control structures into intermediate code.
91. What are the benefits of using intermediate code in a compiler?
92. Describe how expressions are converted into intermediate code.
93. Explain the role of a symbol table in intermediate-code generation.
94. How are arrays and records handled in intermediate-code generation?
95. Discuss the generation of code for boolean expressions and loops.
96. What is the significance of the run-time environment in compiler design?
97. Explain stack allocation of space in run-time environments.
98. How is access to nonlocal data managed on the stack?
99. Describe heap management strategies in run-time environments.
100. Discuss the implementation of dynamic memory allocation and garbage collection.
101. Explain the concept of activation records in the context of run-time environments.
102. How do compilers handle the passing of function arguments at runtime?
103. Describe the role of the heap and stack in memory management.

104. What are the challenges in managing run-time environments for high-level languages?
105. Discuss the impact of run-time environments on the performance of compiled code.
106. How do optimization techniques affect run-time performance?
107. Explain the relationship between intermediate code and machine-specific code generation.
108. What is the role of data flow analysis in optimization?
109. Discuss techniques for optimizing loop performance in compiled code.
110. How are virtual machines used in the context of run-time environments?
111. How does syntax-directed translation influence the efficiency of run-time environments?
112. Discuss the importance of efficient memory management in high-performance computing.
113. Explain the role of intermediate representations in facilitating cross-platform compilation.
114. How do compilers ensure type safety and memory safety during code generation?
115. What are the implications of compiler design on the development of new programming languages?
116. Discuss the role of just-in-time (JIT) compilation in modern run-time environments.
117. How do modern compilers balance between optimization and compilation time?
118. What are the current challenges in automatic memory management?
119. Explain how modern compiler design addresses security concerns.
120. Discuss the future trends in compiler technology and run-time environments.
121. How does the choice of evaluation order in SDDs impact the compiler's efficiency?
122. Describe the advantages and limitations of using three-address code for intermediate representations.
123. In what ways do modern compilers address the challenge of optimizing dynamic data structures at runtime?
124. What strategies are employed by compilers to manage scope and binding of variables in run-time environments?
125. Explain how advancements in compiler design have influenced the development of programming language features.