Short Questions

UNIT - 3

1. What is the primary purpose of intermediate code generation in a compiler?
2. Define syntax tree and its role in intermediate code generation.
3. How does three-address code represent arithmetic expressions?
4. Explain the concept of quadruples in the context of three-address code.
5. What is the difference between static and dynamic type checking?
6. How are types and declarations managed in a compiler?
7. Describe the process of type checking and why it is important in compilers.
8. What are the variants of syntax trees and their significance in compiler design?
9. How is control flow represented in intermediate code?
10. Explain the use of symbol tables in the context of types and declarations.
11. Describe the role of intermediate code in the optimization phase of a compiler.
12. How are switch-statements typically handled in intermediate code generation?
13. What is the significance of directed acyclic graphs (DAGs) in compiler design?
14. How does intermediate code handle procedure calls and returns?
15. Explain the concept of activation records in the context of procedure calls.
16. What is the purpose of using temporary variables in three-address code?
17. How does a compiler ensure type safety during the code generation phase?
18. Discuss the methods used for representing loops in intermediate code.
19. How can intermediate code be targeted for different machine architectures?
20. What strategies are employed for efficient memory management in intermediate code?
21. Explain how arrays are handled in the context of intermediate code generation.
22. Describe the concept of type coercion and its relevance in type checking.
23. How are logical expressions represented in intermediate code?
24. Discuss the challenges of generating intermediate code for complex data structures.
25. What are the benefits of using an intermediate representation in a compiler?

UNIT-4

26. What is the purpose of stack allocation in run-time environments?

27. How is stack allocation implemented in most programming languages?
28. What challenges arise with stack allocation for recursive function calls?
29. How is access to nonlocal data on the stack managed?
30. Explain the role of display or access links in accessing nonlocal data.
31. What is heap management, and why is it necessary?
32. Compare and contrast stack allocation with heap allocation.
33. Describe the basic principle of garbage collection in run-time environments.
34. How does mark-and-sweep algorithm work for garbage collection?
35. What is reference counting in the context of garbage collection?
36. Explain the concept of generational garbage collection.
37. How does tracing garbage collection differ from reference counting?
38. What are the main challenges in implementing garbage collection algorithms?
39. Describe the impact of garbage collection on program performance.
40. Explain trace-based garbage collection and its advantages.
41. What are the key issues in the design of a code generator?
42. How does the choice of target language affect code generation?
43. What is the role of addresses in target code?
44. Explain the concept of basic blocks in code generation.
45. How are flow graphs used in code generation?
46. What strategies are employed for optimization of basic blocks?
47. Describe the process of generating code for a simple code generator.
48. What is peephole optimization, and how does it improve code quality?
49. Discuss the importance of register allocation in code generation.
50. How is register assignment performed in modern compilers?
51. Explain the role of dynamic programming in code generation.
52. How can code generation be optimized for specific target languages?
53. What techniques are used for effective memory management in code generation?
54. How do compilers handle the translation of high-level control structures?
55. What is the impact of optimization techniques on the target code's execution time?
56. Describe the challenges of generating code for parallel and distributed systems.
57. How are live variables analyzed for register allocation?
58. Explain the concept of spill code in register allocation.

59. What role does instruction selection play in code generation?
60. How do compilers ensure the efficient use of hardware resources?
61. Describe the process of liveness analysis in register allocation.
62. What is the significance of loop optimization in code generation?
63. How do global optimization techniques differ from local optimization?
64. Explain the concept of instruction scheduling in code generation.
65. Discuss the challenges in generating code for dynamic languages.
66. How does peephole optimization differ from global optimization strategies?
67. What methods are used for handling array and pointer operations in code generation?
68. Explain the importance of alias analysis in code generation.
69. How do compilers manage the calling conventions in generated code?
70. What strategies are used to minimize the overhead of runtime checks?
71. Describe the role of intermediate representations in code generation.
72. How are complex expressions translated into target code?
73. What is the impact of type information on code generation?
74. Explain how code generation techniques can contribute to security.
75. How does the compiler decide between inline expansion and function calls?

UNIT – 5

76. What is meant by machine-independent optimization in compiler design?
77. List the principal sources of optimization in compiler design.
78. What is the purpose of data-flow analysis in optimization?
79. Define data-flow analysis in the context of compiler optimization.
80. What are the key elements of foundations in data-flow analysis?
81. How does constant propagation contribute to compiler optimization?
82. Explain the concept of partial-redundancy elimination in optimization.
83. What role do loops play in flow graphs within compiler design?
84. How is data-flow analysis used to optimize loops in flow graphs?
85. Describe the process of identifying loops in flow graphs.
86. What is the significance of natural loops in flow graph analysis?
87. How does loop unrolling affect machine-independent optimization?
88. What strategies are employed for loop-invariant code motion?

89. How are induction variables used in compiler optimization?
90. What is dead code elimination, and how is it applied?
91. Describe the concept of code hoisting in compiler optimization.
92. How does strength reduction improve compiler optimization?
93. What is the difference between global and local optimization?
94. How are control-flow graphs utilized in data-flow analysis?
95. Explain the significance of the dominance frontier in compiler design.
96. What techniques are used for constant folding in optimization?
97. How is live variable analysis performed in data-flow analysis?
98. Describe the method of backward data-flow analysis.
99. How does forward data-flow analysis differ from backward data-flow analysis?
100. What is the importance of the control dependence graph in optimization?
101. Explain how reaching definitions are used in optimization.
102. What role does alias analysis play in optimization?
103. How are available expressions identified and utilized in optimization?
104. Describe the concept of value numbering in compiler optimization.
105. How does dependency analysis impact compiler optimization?
106. What is the goal of interprocedural optimization in compilers?
107. How are escape analysis and its implications for optimization?
108. Describe how optimization affects register allocation strategies.
109. What challenges are presented by dynamic aliasing in optimization?
110. How do compilers perform range analysis for optimization?
111. Explain the role of loop fusion in optimization.
112. What is loop distribution, and how does it contribute to optimization?
113. How is speculative execution used in compiler optimization?
114. What is the impact of inlining functions on optimization?
115. How do compilers detect and optimize tail recursion?
116. Describe the process and benefits of array bounds checking optimization.
117. How is exception handling optimized in modern compilers?
118. What is the role of profile-guided optimization in compiler design?
119. How do compilers optimize memory access patterns?
120. What strategies are used for optimizing recursive function calls?

121. How does the compiler optimize conditional branches?
122. What is the significance of SSA (Static Single Assignment) form in optimization?
123. How are side effects managed during optimization processes?
124. Explain the concept of aggressive dead code elimination.
125. How do compilers handle optimization in the presence of volatile variables?