Compiler Design

SECTION A: Short Questions

1. What is a translator?
2. Differentiate between a compiler and an assembler.
3. Discuss the conversion of NFA into DFA and give the algorithm used.
4. Write a short note on the symbol table.
5. Describe the data structure for a symbol table.
6. What is an activation record?
7. What is postfix notation?
8. Define Three Address Code.
9. What are quadruples?
10. What do you mean by regular expression?
11. What is YACC? Discuss it.
12. Design a DFA for the regular expression: (x+y)*xyy.
13. Compute FOLLOW(B) for the grammar: S→B|SabS, B→bB|ε.
14. Discuss shift-reduce parsing.
15. Find the postfix notation for the expression: (a+b+c)*(c+q).
16. Discuss non-linear type intermediate code.
17. Write a short note on "Activation Record."
18. Discuss hash tables.
19. Discuss constant folding.
20. Discuss the designing issues of a code generator.
21. What is the difference between a parse tree and an abstract syntax tree?
22. Explain the problems associated with top-down parsers.
23. What are the various errors that may appear in the compilation process?
24. What are the two types of attributes associated with a grammar symbol?
25. Define the terms Language Translator and Compiler.
26. What is hashing? Explain.
27. What do you mean by left factoring grammars? Explain.
28. Define left recursion. Is the grammar E→E+E|E*E|a|b left recursive?
29. What is an ambiguous grammar? Give an example.
30. List the conflicts during shift-reduce parsing.

31. How will you group the phases of a compiler?
32. Mention the role of semantic analysis.
33. What are the various parts of a LEX program?
34. Differentiate between parse tree and syntax tree with an example.
35. Give the properties of intermediate representation.
36. Differentiate between LR and LL parsers.
37. What is phrase-level error recovery?
38. Discuss the capabilities of CFG.
39. Define loop jamming.
40. What is an induction variable?
41. Define bootstrapping in the context of compilers.
42. Which phase of the compiler is optional and why?
43. Explain the concept of shift-reduce parsing.
44. Define syntax-directed translation schemes.
45. Discuss how scope information is represented in a symbol table.
46. Discuss two design issues in code generation.
47. Explain the concept of global data-flow analysis.

SECTION B: Descriptive Questions

1. Write regular expressions for:
    - The set of all strings over {a,b} such that the fifth symbol from the right is 'a'.
    - The set of all strings over {a,b} such that every block of four consecutive symbols contains at least two '0's.
2. Construct an NFA for the regular expression a/abb/a*b* using Thompson's construction methodology.
3. Eliminate left recursion from the grammar: S→AB, A→BS|b, B→SA|a.
4. Explain non-recursive predictive parsing. Construct the predictive parsing table for the grammar: E→TE', E'→+TE'|ε, T→FT', T'→*FT'*/*ε, F→F*|a|b.
5. Explain the process of compilation for the statement: a = b + c * / 0.
6. Construct the CLR(1) parsing table for the grammar: S→AA, A→aA|b.
7. Give syntax-directed definitions to construct a parse tree for the input expression 4*/+3*9.
8. Explain lexical, syntax, and semantic phase errors in detail.
9. Explain loop optimization in detail.
10. Construct the LALR parsing table for the grammar: S→BB, B→aB|b.
11. Explain how an activation record is related to runtime storage organization.
12. Write quadruple, triple, and indirect triple representations for the expression: (x + y)*(y + z) + (x + y + z).
13. Discuss:

- Basic block
- Next-use information
- Flow graph

14. Construct a predictive parse table for the grammar: E→E+T|T, T→T*F|F, F→F/a|b.
15. Describe the relationship between finite state machines and regular expressions.
16. Construct the LR(1) and LALR parsing tables for the grammar: S→aAd|bBd|aBe|bAe, A→f, B→f.
17. Explain quadruples and triples in syntax-directed translation.
18. Describe a stack allocation scheme for managing memory during program execution.
19. Explain the role of a code generator in a compiler.
20. Write SDD to produce three-address code for Boolean expressions.
21. Discuss stack allocation and heap allocation strategies with examples.
22. Explain attributed grammars and translation schemes for infix to postfix conversion.
23. Construct NFA and DFA for the regular expression: (0+1)*(00+11)(0+1)*.
24. Explain lexical and syntax analysis phases with error reporting.

SECTION C: Long Answer Questions

1. Give the Operator-Precedence parsing algorithm. Build the operator precedence table for the grammar:
   E→E+T|T, T→T*F*|*F, F→(E)|id. Parse the input string (id+(id*id)).*
2. Construct the LR(1) parsing table for the grammar: S→aAd|bBd|aBe|bAe, A→f, B→f. Draw the LALR table.
3. Translate (C+D)*(E+Y) into postfix using syntax-directed translation schemes (SDTS).
4. Construct the SLR parsing table for the grammar: S→0S0|1S1|10.
5. Explain logical and syntactic phase errors and suggest recovery methods.
6. Generate three-address code for: C[A[i,j]] = B[i,j] + C[A[i,j]] + D[i+j].
7. Give algorithms for eliminating local and global common subexpressions.
8. Optimize the given three-address code using function-preserving transformations and loop optimization.
9. Write short notes on:
   - Loop optimization
   - Global data analysis
   - Direct acyclic graph
   - YACC parser generator
10. Construct the SLR parse table for the grammar: E→E+E|E*E|id.
11. Differentiate between stack allocation and heap allocation.
12. Write syntax-directed definitions for assignment statements.
13. Explain the advantages of DAG and peephole optimization.
14. Explain lexical and syntactic errors and recovery methods.
15. Detect and eliminate induction variables from the given intermediate code.
16. Test whether the grammar is LL(1) and construct the parsing table.
17. Distinguish between static and dynamic scope. Explain access to non-local names in static scope.
18. Discuss design issues in code generators and loop optimization.
19. Generate three-address code for a while loop with nested if-else.
20. Construct the CLR parse table for the grammar: A→BB, B→cB|d.
21. Construct the SLR parsing table for the grammar: S→0S0|1S1|10.
22. Explain backpatching and generate three-address code for Boolean expressions.
23. Explain top-down parsing and its problems with examples.
24. Draw and explain the general activation record structure.
25. Explain scope representation using scope by number and scope by location.

26. Define symbol table and explain its data structures.
27. Explain:

- Copy propagation
- Dead-code elimination
- Code motion
- Reduction in strength

28. Explain DAG representation of a basic block with an example.
29. Write quadruple, triple, and indirect triple representations for: a = b * –c + b * –c.
30. Construct an NFA for the regular expression: a(b|c)*.
31. Check if the grammar E→E+E|E*E|id is ambiguous and convert it to unambiguous.
32. Check if the grammar S→PQy, P→Sy|x, Q→yS is LR(0).
33. Apply shift-reduce parsing to construct a parse tree for id * (id+id).
34. Explain syntax-directed translation for array references in arithmetic expressions.
35. Define semantic errors and discuss their detection challenges.
36. Apply common subexpression elimination to optimize a basic block.
37. Construct a DAG for the basic block: x = a + b, y = c – d, z = x * y.