GOVERNMENT ENGINEERING COLLEGE-MODASA	
Question Bank	
Faculty	Mr. C. H. Makwana
Subject Name	Compiler Design
Subject Code	2170701
Semester	7th
Department	Computer Engineering
Term	ODD 2016

Chapter 1 to 4:

1. Explain the phases of compiler with an example. (Summer 2016) (Summer 2015)
2. List out phases of a compiles. Write a brief not on Lexical Analyzer. (Summer 2014)
3. Define lexeme, token and pattern. Identify the lexemes that make up the tokens in the following program segment. Indicate corresponding token and pattern.

```
void swap (int a, int b)
{
        int k;
        k = a;
        a = b;
        b = k;
}
```
(Winter 2015)
4. Explain Semantic Analysis and Syntax Analysis phases of compiler with suitable example. Also explain the reporting errors by these two phases. (Winter 2015)
5. What is the use of a symbol table? How the identifiers are stored in the symbol table? (Summer 2014) (Winter 2014) (Winter 2015)
6. What is the pass of a compiler? Explain how the single and multi-pass compilers work. (Summer 2014)
7. What is regular expression, give all the algebraic properties of regular expression. (Summer 2016)
8. Unsigned numbers are strings such as 5280, 39.37, 6.336E4 or 1.894E-4, give the regular definitions for the above mentioned strings. (Summer 2016)
9. What are regular expressions? Find the regular expression described by DFA {{A,B},{0,1},δ,A,{B}},where δ is detailed in following table .

	0	1
A	A	B
B	φ	A

Please note B is accepting state.

Describe the language defined by the regular expression. (Summer 2015)
10. Draw the state transition diagram for the unsigned numbers. (Summer 2016)
11. Draw Transition diagram of following:
    i. relational operators.
    ii. unsigned operator.
12. Draw Deterministic Finite Automata for :
    (0+1)*101(0+1)*
    10(0+1)*1 (Summer 2014)
13. Convert the (a|b|c)*d*(a*|b)ac+# regular expression to DFA directly and draw its DFA. (Summer 2016)

14. Construct DFA for following Regular expression. Use firstpos, lastpos and followpos functions to construct DFA.
    (a * | b *) * (Winter 2015)
15. Construct minimum state DFA's for following regular expressions. i. (a|b)*a(a|b) ii. (a|b)*a(a|b) (a|b)
16. Construct DFA by syntax tree construction method.

    $a^+ b^* (c|d)\ f\ \#$

    Optimize the resultant DFA. (Summer 2015)
17. Draw the DFA for the regular expression (a|b)*abb using set construction method only. (Summer 2016)
18. Construct NFA for following Regular Expression using Thomson's Construction. Apply subset construction method to convert into DFA.
    (a | b)*abb (Winter 2015)
19. Construct the NFA using thompson's notation for following regular expression and then convert it to DFA.

    $a^+ (c|d)\ b^* f\ \#$ (Summer 2015)

20. Write a brief note on input buffering techniques. (Summer 2014)
21. Explain Buffer pairs and Sentinels.(Winter 2014)

22. Define: Left Recursive. State the rule to remove left recursive from the grammar. Eliminate left recursive from following grammar.
    S -> Aa | b
    A -> Ac | Sd | f (Winter 2015)
23. What is left recursion? Eliminate the left recursion from the following grammar.
    E -> E + T | T
    T ->T * F | F
    F ->(E) | id (Summer 2016)
24. How top down and bottom up parser will parse the string **'bbd'** using grammar A -> bA | d. Show all steps clearly.(Summer 2015)
25. Implement the following grammar using Recursive Descent Parser.
    S -> Aa | bAc | bBa, A -> d, B -> d (Summer 2014)
26. Design the FIRST SET and FOLLOW SET for the following grammar.
    E -> E + T | T
    T ->T * F | F
    F ->(E) | id (Summer 2016)
27. Construct LL(1) parsing table for the following Grammar:
    S -> (L) | a
    L -> L , S | S (Winter 2015)
28. Implement the following grammar using Table Driven parser and check whether it is LL(1) or not. S -> aBDh, B -> cC, C -> bC / ^, D -> EF, E -> g / ^, F-> f / ^ (Summer 2014)
29. Develop a predictive parser for the following grammar.
    S'->S
    S->aA|b|cB|d
    A->aA|b
    B->cB|d (Summer 2015)
30. For the following grammar

D -> T L ;

L -> L , id | id

T -> int | float

1)Remove left recursion (if required)

2)Find first and follow for each non terminal for Resultant grammar

3)Construct LL(1) parsing table

4)Parse the following string (show stack actions clearly) and draw parse tree for the input:

int id, id; (Summer 2015)

31. Explain recursive-descent and predictive parsing. (Winter 2014)
32. What do you understand by a handle? Explain the stack implementation of shift reduce parser with the help of example. (Winter 2014)
33. What is bottom-up parsing? Discuss Shift Reduce parsing technique in brief. What is a handle? (Summer 2014)
34. Given the Grammar, evaluate the string id-id*id using shift reduce parser.
    E-> E – E
    E -> E * E
    E -> id
35. Construct the collection of sets of LR(0) items for the following grammar.
    S-> Aa | bAc | dc | bda
    A->d (Summer 2015)
36. Explain SLR parser in detail with the help of a suitable example. (Summer 2016)
37. Construct SLR Parsing Table for the following grammar.
    S->0S0 | 1S1 | 10 (Winter 2015)
38. Construct an SLR Parsing table for the following grammar.
    E->E-T|T
    T->F↑T|F

    F->(E) | id (Summer 2015)

39. Construct SLR parsing table for the following grammar :
    E->E+T E->T T->T*F T->F F->(E) F->a (Summer 2014)
40. Check whether the following grammar is CLR or not.
    S -> Aa | bAc | Bc | bBa
    A -> d
    B ->d (Winter 2015)
41. Explain LALR parser in detail. Support your answer with example. (Winter 2015)
42. Differentiate SLR, Canonical LR and LALR. Also justify the statement "A class of grammar that can be parsed using LR methods is a proper subset of the class of grammars that can be parsed with predictive parser". (Summer 2016)
43. Show that the following grammer S-> AaAb | BbBa A -> ϵ B -> ϵ is LL(1) but not SLR(1). (Winter 2014)
44. Show that the following grammer S->Aa | bAc | dc | bda A->d is LALR(1) but not SLR(1). (Winter 2014)
45. Show that the following grammer
    S->Aa | bAc | Bc | bBa

A->d

B->d is LR(1) but not LALR(1). (Winter 2014)

46. Explain Operator Precedence Parsing method with example. (Winter 2015)
47. Where do we use operator precedence parsing technique? Give the general precedence table for operating precedence parsing, considering all the generalized rules. (Summer 2016)
48. Define an Operator Precedence Grammar. Also write down the rules to find relationship between each pair of terminal symbols. (Summer 2014)
49. Compare top-down and bottom-up parser
50. Write short note on context free grammar (CFG) explain it using suitable example. (Summer 2016)
51. Write a context free grammar for arithmetic expressions. Develop a syntax directed definition for the grammar. Draw an annotated parse tree for the input expression: **(3*2+2)*4** (Summer 2015)
52. Show syntax directed definition for simple desk calculator. Also show annotated parse tree for 3*5+4n, where n indicates newline. (Winter 2015)
53. Write a syntax directed definition of a simple desk calculator and draw an annotated parse tree for 4*3 + 2*5 n. (Summer 2014)
54. Construct a Syntax-Directed Translation scheme that translates arithmetic expressions from infix into postfix notation. Show the application of your scheme to the string "3*4+5*2". (Winter 2014)
55. Explain with an appropriate example how to perform bottom up evaluation of an inherited attributes. (Winter 2014)
56. Discuss synthesized and inherited attributes using a suitable grammar. (Summer 2015)
57. Differentiate Synthesized and Inherited attributes. (Summer 2014)
58. Explain all error recovery strategies of compiler using suitable examples. (Summer 2016) (Summer 2014) (Winter 2014)
59. List the errors generated by the syntax analysis phase. Discuss error handling methods in the syntax analysis phase. (Summer 2015)

Chapter 5 to 8:

60. What is intermediate code? What is its importance? Discuss various representations of three address code. (Summer 2015)
61. Explain quadruple, triple and indirect triple with suitable example(Winter 2014).
62. Translate the expression – (a*b)+(c*d)+(a*b*c) into
    1. Quadruples
    2. Triples
    3. Indirect triples. (Summer 2016)
63. Translate following arithmetic expression
    - (a * b) + (c + d) - (a + b + c + d) into
    1] Quadruples
    2] Triple
    3] Indirect Triple (Winter 2015)
64. Convert the following statement into triple, indirect triple and quadruple forms.
    A= (B+C) $ E + (B+C) *F (Summer 2015)
65. Convert the following into quadruple, triple and indirect triple forms : -(a+b)*(c-d).
66. Explain various dynamic storage allocation techniques. (Winter 2015)
67. Write a note on static and dynamic memory allocation. What do you mean by dangling reference? (Summer 2014)

68. Explain Stack allocation and Activation record organization in brief. (Winter 2014)
69. What is an activation record? Explain how they are used to access local and global variables. (Summer 2016) (Summer 2014)
70. Elaborate the term "Activation Record" in detail. (Summer 2015)
71. Discuss various code optimization techniques. (Winter 2015) (Summer 2015) (Winter 2014)
72. Describe algorithm for global common subexpression elimination.(Winter 2014)
73. Explain various issues in design of code generator. (Winter 2015) (Winter 2014)
74. Explain how type checking & error reporting is performed in compiler.
    Draw syntax tree and DAG for the statement
    a = (a * b + c) ^ (b + c) * b + c. Write three address codes from both. (Summer 2016)
75. Explain peephole optimization. (Summer 2016) (Summer 2014)

76. Explain the following:
    1) The Handle
    2) Left Factoring
    3) Directed Acyclic Graph
    4) Conflicts in LR Parsing
    5) Parser Generator
    6) Dependency Graph
    7) Locality of reference (Summer 2015)

Note: This is not your Final assignment. These questions is of old university papers and it will be helpful to you during your mid semester exam and final exam.