QUESTION BANK

MODULE 1

1. Describe input buffering scheme in lexical analyzer. **(KTU, MAY 2019)**
2. Construct a regular expression to denote a language L over ∑= {0,1} accepting all strings of 0's and 1's that do not contain substring 011 **(KTU, MAY 2019)**
3. Develop a lexical analyzer for the token identifier. **(KTU, MAY 2019)**
4. Scanning of source code in compilers can be speeded up using input buffering. Explain. **(KTU, DECEMBER 2019)**
5. Draw the DFA for the regular expression (a | b)* (abb | a+ b). **(KTU, DECEMBER 2019)**
6. Explain compiler writing tools. **(KTU, DECEMBER 2019) (KTU, MAY 2019)**
7. Explain how the regular expressions and finite state automata are used for the specification and recognition of tokens? **(KTU, DECEMBER 2019)**
8. Explain the working of different phases of a compiler. Illustrate with a source language statement. **(KTU, DECEMBER 2019) (KTU, MAY 2019)**
9. Describe input buffering scheme in lexical analyzer.
10. Draw the transition diagram for the regular definition,

    relop➔< I <= I = I <> I >= I > **(KTU, APRIL 2018)**
11. With an example source language statement, explain tokens, lexemes and patterns. **(KTU, APRIL 2018)**
12. Apply bootstrapping to develop a compiler for a new high level language P on machine N. **(KTU, APRIL 2018)**
13. Now I have a compiler for P on machine N. Apply bootstrapping to obtain a compiler for P on machine M. **(KTU, APRIL 2018)**
14. Define cross-compilers. **(KTU, APRIL 2018)**
15. For a source language statement, a= b*c - 2, where a, b and c are float variables, * and - represents multiplication and subtraction on same data types, show the input and output at each of the compiler phases. **(KTU, APRIL 2018)**
16. Write a short note on compiler construction tools. **(KERALA, MARCH 2015), (CALICUT, APRIL 2017),(CALICUT, APRIL 2017), (CALICUT, APRIL 2015)**
17. Explain the role of lexical analyser. **(KERALA, MARCH 2015), (CALICUT, JUNE 2010)**
18. Explain structure and different phases of compiler. **(KERALA, MARCH 2015)**
19. Explain Bootstrapping with example. **(KERALA, APRIL 2014)**
20. Write a short note on error handling. **(KERALA, APRIL 2014)**
21. Discuss the role of symbol table in compiler design process. **(CALICUT, APRIL 2017)**

MODULE 2

1. Find the FIRST and FOLLOW of the non-terminals in the grammar

   S→aABe

   A→Abc|b

   B→d **(KTU, MAY 2019)**

2. Consider the context free grammar **S→1aSbS | bSaS | €** Check whether the grammar is ambiguous or not. **(KTU, MAY 2019)**
3. What is Recursive Descent parsing? List the problems faced in designing such a parser. **(KTU, MAY 2019)**
4. Design a recursive descent parser for the grammar

   E→E + T | T

   T→T*F | F

   F→(E) | id **(KTU, MAY 2019)**

5. What is left recursive grammar? Give an example. What are the steps in removing left recursion? **(KTU, MAY 2019)**
6. Differentiate leftmost derivation and rightmost derivation. Show an example for each. **(KTU, DECEMBER 2019)**
7. Find out context free language for the grammar given below:

   S → abB

   A → aaBb | ε

   B → bbAa **(KTU, DECEMBER 2019)**

8. Given a grammar :

   S→ (L)|a

   L→ L,S | S

   Is the grammar ambiguous? Justify. **(KTU, DECEMBER 2019)**

9. Give the parse tree for the string **(a,((a,a), (a,a)))** **(KTU, DECEMBER 2019)**
10. Can recursive descent parsers used for left recursive grammars? Justify your answer. Give the steps in elimination of left recursion in a grammar. **(KTU, DECEMBER 2019)**
11. Compute FIRST and FOLLOW for the grammar: **S→ SS+ | SS* | a** **(KTU, DECEMBER 2019)**
12. Construct the predictive parsing table for the following grammar:

    S → (L) | a

    L → L,S | S **(KTU, DECEMBER 2019)**

13. Define **LL(1)** grammars. **(KTU, APRIL 2018)**
14. Is the grammar **S➔S(S)S/ε** ambiguous? Justify your answer. **(KTU, APRIL 2018)**
15. Design a recursive descent parser for the grammar **S➔cAd, A➔ab/ b** **(KTU APRIL 2018)**

16. Consider the following grammar

E→ E or T |T

T→ T and F |F

F → not F | (E) | true | false

(i) Remove left recursion from the grammar.

(ii) Construct a predictive parsing table.

(iii) Justify the statement " The grammar is LL (1)". **(KTU, APRIL 2018)**

17. Compute the FIRST and FOLLOW for the following Grammar.

S→Bb/Cd

B→aB/ε

C→cC/ε **(KTU, APRIL 2018)**

18. Compute the FIRST and FOLLOW set of A in the following grammar

A→(A) A|ε **(KERALA, MARCH 2015)**

19. Transforms the following grammar so that it will be LL(1) without changing the language.

S→aAC|bB

A→Abc|Abd|ε

B→f|g

C→h|i **(KERALA, MAY 2011)**

20. Define parse tree and syntax tree **(KERALA, APRIL 2014), (CALICUT, APRIL 2016)**

21. Explain drawbacks of top down parsing. **(KERALA, APRIL 2014)**

22. What is left recursion? How is eliminated? Explain with examples. **(CALICUT, JUNE 2010)**

23. Explain the stack implementation of shift reduce parsing technique with suitable examples. **(CALICUT, JUNE 2010)**

24. Consider following grammar

S→AaAb|BbBb

A→ ε

B → ε

(i) Construct predictive parser table

(ii) Is it LL(1) parser **(CALICUT, APRIL 2017)**

MODULE 3

1. Explain operator grammar and operator precedence parsing.**(KTU, MAY 2019)**

2. Find the LR(0) items for the grammar **S→SS | a| €.** **(KTU, MAY 2019)**

3. Derive LALR (1) parsing algorithm for following grammar

S→AS/b

A→SA/a **(KTU, MAY 2019)**

4. Explain the main actions in a shift reduce parser. **(KTU, MAY 2019)**
5. What are different parsing conflicts in SLR parsing table? **(KTU, MAY 2019)**
6. Write the algorithm to construct LR(1) collection for a grammar. **(KTU, DECEMBER 2019)**
7. Write algorithm for SLR paring table construction. **(KTU, DECEMBER 2019)**
8. Construct the SLR table for the grammar: **S → aSbS | a** **(KTU, DECEMBER 2019)**
9. Differentiate CLR and LALR parsers. **(KTU, DECEMBER 2019)**
10. Construct canonical LR(0) collection of items for the grammar below.

    S➔ L=R
    S➔R
    L ➔ *R
    L ➔ id
    R ➔ L

    Also identify a shift reduce conflict in the LR(0) collection constructed above. **(KTU, APRIL 2018), (KERALA, APRIL 2014)**

11. Demonstrate the identification of handles in operator precedence parsing? **(KTU, APRIL 2018)**
12. Construct L ALR parse table for the grammar **S→CC, C→cC|d** **(KTU, APRIL 2018)**
13. List all the LR(0) items for the grammar **S→AS|b , A→SA|a** **(KERALA, MARCH 2015)**
14. Explain bottom up parsing. **(KERALA, MARCH 2015)**
15. Define operator grammar with example. **(KERALA, APRIL 2014)**
16. Construct a CLR parsing table for the following grammar

    S→Ba|bBc|dc|bda
    B→d

    Write the algorithm for the construction of ACTION and GOTO table **(KERALA, MAY 2011)**

17. Compare LR Canonical LR and LALR parsers. **(KERALA, APRIL 2014)**

MODULE 4

1. Design a type checker for simple arithmetic operations. **(KTU, MAY 2019)**
2. Explain the syntax directed definition of a simple desk calculator. **(KTU,MAY 2019)**
3. What is an SDD? Show an example. **(KTU, DECEMBER 2019)**
4. Give the annotated parse tree for the expression: **1*2*3*(4+5)n** **(KTU, DECEMBER 2019)**

5. Distinguish between synthesized and inherited attributes. **(KTU, DECEMBER 2019)**
6. Construct syntax directed translation scheme for infix to postfix translation. **(KTU, DECEMBER 2019)**
7. Explain the specification of a simple type checker. **(KTU, DECEMBER 2019)**
8. Design a Syntax Directed Definition for a Desk calculator that prints the result. **(KTU, APRIL 2018), (CALICUT, JUNE 2010)**
9. Describe the type checking of functions. **(KTU, APRIL 2018)**
10. Define S-attributed and L -attributed definitions. Give an example each. **(KTU, APRIL 2018), (KTU, MAY 2019), (CALICUT, APRIL 2017), (CALICUT, APRIL 2016)**
11. Explain bottom- up evaluation of S- attributed definitions. **(KTU, APRIL 2018), (KTU, MAY 2019)**
12. With an SDD for a desk calculator, give the appropriate code to be executed at each reduction in the LR parser designed for the calculator. Also give the annotated parse tree for the expression **(3 * 5) - 2** **(KTU, APRIL 2018)**
13. Consider the following grammar

    S→ En
    E→E+T|T
    T→T*F|F
    F→(E)|id

    Draw an annotated parse tree for the expression **7*9+5**
14. Perform the bottom up evaluation and show the result **(CALICUT, APRIL 2015)**
15. What is annotated parse tree? Give example **(KERALA, MAY 2011), (KTU, MAY 2019)**
16. Write the syntax directed translation scheme for a desk calculator and give the parse tree (with translations) for the input **(24 + 51) * 9 + 17** **(KERALA, APRIL 2014)**
17. Define synthesized and inherited translations. **(KERALA, APRIL 2014)**
18. Explain the process of type checking and type conversion done for arithmetic operations **(CALICUT, JUNE 2010)**

MODULE 5

1. Explain quadruples, triples and dags with an example each. **(KTU, MAY 2019)**
2. Explain the principal sources of optimization **(KTU, MAY 2019)**
3. Explain optimization of basic blocks. **(KTU, MAY 2019)**
4. With suitable examples explain loop optimization. **(KTU, MAY 2019)**
5. Explain issues in design of a code generator **(KTU, MAY 2019)**
6. Explain simple code generation algorithm. **(KTU, MAY 2019)**

7. Explain intermediate code generation of an assignment statement **(KTU, MAY 2019)**
8. Explain storage organization and storage allocation strategies. **(KTU, MAY 2019)**
9. Explain how DAGs help in intermediate code generation? **(KTU, DECEMBER 2019)**
10. Define the following and show an example for each.

    Three-address code
    Triples
    Quadruples
    Indirect triples **(KTU, DECEMBER 2019)**
11. How is storage organization and management done during runtime? **(KTU, DECEMBER 2019)**
12. Write the algorithm for partitioning a sequence of three-address instructions into basic blocks. **(KTU, DECEMBER 2019)**
13. Construct the DAG and three address code for the expression

    a+a*(b-c)+(b-c)*d **(KTU, DECEMBER 2019)**
14. Write syntax directed definitions to construct syntax tree and three address code for assignment statements. **(KTU, APRIL 2018)**
15. Explain quadruples and triples with an example each. **(KTU, APRIL 2018)**
16. Construct the syntax tree and then draw the DAG for the statement

    e:= (a*b) + (c-d) * (a*b) **(KTU, APRIL 2018)**
17. Explain static allocation and heap allocation strategies. **(KTU, APRIL 2018)**
18. Write quadruples for the expression **(a+b) * (c+d) - (a+b+c)** **(KERALA, MARCH 2015)**
19. Show that the following grammar is LR(1) but note LALR(1)

    S→Aa|bAC|Bc|bBa
    A→d
    B→d **(KERALA, MARCH 2015)**
20. Write a translation scheme to implement Boolean expression. **(KERALA, MARCH 2015), (KERALA, MAY 2011)**
21. Write notes on translation of assignment statement into three address code. **(KERALA, MARCH 2015), (KERALA, APRIL 2014)**
22. Write a note on quadruples. **(KERALA, APRIL 2014)**
23. Write a note on triples and indirect triples with examples. **(KERALA, APRIL 2014)**
24. Construct the Directed Acyclic Graph for the basic block given below and simplify the three address code

    d = b * c
    e = a + b
    b = b * c
    a = e – d **(CALICUT, APRIL 2016), (CALICUT, APRIL 2015)**

25. Discuss the storage allocation strategies. **(CALICUT, APRIL 2017) (CALICUT, JUNE 2010)**

26. What are the different ways to representing intermediate codes? Give examples. **(CALICUT, APRIL 2016)**

MODULE 6

1. Explain the code generation algorithm. Illustrate with an example. **(KTU, DECEMBER 2019)**

2. State the issues in design of a code generator. **(KTU, DECEMBER 2019)**

3. Explain different code optimization techniques available in local and global optimizations? **(KTU, DECEMBER 2019)**

4. How the optimization of basic blocks is done by a compiler? **(KTU, DECEMBER 2019)**

5. With an example each explain the following loop optimization techniques:
Code motion
Induction variable elimination and
strength reduction **(KTU, APRIL 2018)**

6. Explain any two issues in the design of a code generator. **(KTU, APRIL 2018) (CALICUT, APRIL 2015)**

7. Explain the optimization of basic blocks. **(KTU, APRIL 2018)**

8. Write the Code Generation Algorithm and explain the getreg function. **(KTU, APRIL 2018)**

9. Generate a code sequence for the assignment **d=(a-b)+(a-c)+(a-c)** **(KTU, APRIL 2018)**

10. Discuss on loop optimization. **(KERALA, MARCH 2015), (KERALA, APRIL 2014)**

11. Explain the operation of a simple code generator for pointer assignments and conditional statements. **(KERALA, MARCH 2015)**

12. Discuss the issues in design of a code generator. **(CALICUT, APRIL 2017)**

13. What is the use of algebraic identities in optimization of basic blocks? **(CALICUT, APRIL 2016)**

14. What is a flow graph? Explain the process of constructing flow graphs and the use of flow graphs in code optimization. **(CALICUT, DECEMBER 2010)**

****