Reg No.:_____ Name:_____

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

B.Tech Degree S6 (R,S) (FT/WP/PT) Examinations April 2026 (2019 Scheme)

Course Code: CST302

Course Name: COMPILER DESIGN

Max. Marks: 100 Duration: 3 Hours

PART A

Answer all questions, each carries 3 marks. Marks

1	Discuss different roles of lexical analyser. Identify the lexemes and its corresponding tokens in the statement **p=q+a*100**	(3)
2	Explain any three compiler construction tools.	(3)
3	Can recursive descent parser be used for left recursive grammars? Justify your answer with an example.	(3)
4	Compute FIRST and FOLLOW for the following grammar S→Bb/Cd B→aB/Ɛ C→cC/Ɛ	(3)
5	Demonstrate the identification of handles in operator precedence parsing.	(3)
6	Describe handle pruning with the help of an example.	(3)
7	Write the SDD for a simple type declaration and draw the annotated parse tree for the declaration **float a, b, c**.	(3)
8	What is activation record? Describe its structure.	(3)
9	Explain basic block and program flow graph with example.	(3)
10	Explain any two machine independent code optimization techniques	(3)

PART B

Answer one full question from each module, each carries 14 marks.

Module I

11	a)	Discuss the different phases of compiler with neat diagram. Trace the output after each phase of the compiler for the assignment statement result = first + last * 90 Assume all variables given are of float type.	(8)
	b)	Discuss the relevance of input buffering scheme in lexical analyzer. Also mention	(6)

the importance of sentinels in input buffering.

OR

12 a) Explain how lexical analyser recognizes keywords, identifiers, relational operators, and numbers in the source program. (9)

b) Discuss the concept of bootstrapping in compiler design with an example. (5)

Module II

13 a) Remove left recursion from the following grammar and construct predictive parsing table (8)

E→ E or T | T

T→ T and F | F

F→ not F | (E) | true| false

b) Design a recursive descent parser for the grammar (6)

E->E + T | T

T->T*F | F

F->(E) | id

Also show how the parser works for the input string id*id+id

OR

14 a) Prove that the following grammar is not LL(1) (8)

S →iEtSS' | a

S → eS | ε

E → b

b) What is an ambiguous grammar? Check whether the given grammar is ambiguous or not (3)

S →(L)|a

L → L,S | S

c) Perform left factoring for the following grammar (3)

S → iEtS / iEtSeS / a

E → b

Module III

15 a) Construct SLR(0) parsing table for the following grammar (9)

$T \rightarrow M = S \mid S$

$M \rightarrow * S \mid id$

$S \rightarrow M$

b) Explain the algorithm of operator precedence parser. (5)

OR

16 a) Construct CLR(1) parsing table for the grammar (8)

$W \rightarrow QQ$

$Q \rightarrow qQ \mid x$

b) Construct canonical LR (0) collection of items for the grammar below. (6)

$S \rightarrow L = R$

$S \rightarrow R$

$L \rightarrow * R$

$L \rightarrow id$

$R \rightarrow L$

Also check for shift-reduce or reduce-reduce conflict in the LR(0) collection constructed above.

Module IV

17 a) Differentiate S attributed and L attributed definitions. Give the S-attributed SDD of a simple desk calculator and show annotated parse tree for the expression (3+4)*(5+6). (9)

b) Discuss the bottom-up evaluation of S-attributed definitions. Explain with an example (5)

OR

18 a) Construct the syntax tree and then draw the DAG for the statement e = (a*b) - (c-d) * (a*b) . Write the three-address code sequence, quadruple and triple representation of DAG for above expression. (8)

b) Explain stack and heap memory allocation strategies. (6)

Module V

19 a) Explain local and global optimization with relevant example (7)

b) Discuss code generation in detail, highlighting issues in design of code generator and a simple target language. (7)

OR

20 a) Discuss any two optimization strategy with example. (6)

b) Give the code generation algorithm and explain the *getreg* function. Using this algorithm generate the code for the expression z = (x+y) + (a-b) (8)

***