USN | | | | | | | | | | |

MANGALORE INSTITUTE OF TECHNOLOGY & ENGINEERING

(A Unit of Rajalaxmi Education Trust®, Mangalore)
Autonomous Institute affiliated to VTU, Belagavi, Approved by AICTE, New Delhi
Accredited by NAAC with A+ Grade & ISO 9001:2015 Certified Institution

Model Question Paper

Sixth Semester B.E Degree Examination

Compiler Design

Time: 2.5 Hours (150 Minutes) **Max. Marks: 100**

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M: Marks, L: RBT (Revised Bloom's Taxonomy) level, C: Course outcomes.

		Module -1	**M**	**L**	**C**
Q1	a.	Show the step-by-step working of language processors using a neat block diagram.	6	L2	CO1
	b.	Illustrate the working of single-buffer and double-buffer schemes with neat diagrams and demonstrate why the double-buffer scheme is preferred in compiler implementation.	6	L2	CO2
	c.	Illustrate and explain the structure of a compiler with a neat diagram, and demonstrate the function of each phase.	8	L2	CO1
		OR			
Q2	a.	Explain error recovery mechanisms to handle lexical errors in a given source program and illustrate with an example.	6	L2	CO1
	b.	Draw the transition diagrams for recognizing: i) relop ii) Unsigned number	6	L2	CO2
	c.	Explain the structure of a compiler and Demonstrate the working of each phase of the compiler for the input statement position = initial + rate * 60 and show the output produced at every phase.	8	L2	CO1
		Module-2			
Q3	a.	Consider the grammar: E → E + T | T T → T * F | F F → (E) | id a) Eliminate left recursion and left factor the grammar wherever required. b) Compute FIRST and FOLLOW sets for the transformed grammar.	10	L3	CO3
	b	Construct LL(1) parsing for the grammar given below .Show moves made by predictive for the string id+id*id E->TE' E' ->+TE' T->FT' T'->*FT' F-> (E) | id	10	L3	CO3
		OR			
Q4	a.	Construct Canonical LR(0) items for the grammar given below S->L=R L -> *R | id R -> L	10	L3	CO3
	b.	Consider the grammar:	10	L3	CO3

		E → E+T/T T → TF/F F → F* | a | b Construct SLR parsing table.			
Module – 3					
Q5	a.	Define synthesized attribute. Develop a Syntax Directed Definition (SDD) for arithmetic expressions involving '+' and '*' and demonstrate its application by evaluating a given expression. Evaluate the expression 3 + 4 * 5.	10	L3	CO2
	b.	Construct Syntax Directed Definition (SDD) for simple type declarations and draw dependency graph for a declaration **float id_1, id_2,id_3 .**	10	L3	CO2
OR					
Q6	a.	Define inherited attribute. Develop a Syntax Directed Definition (SDD) for **T -> FT`** **T`->* FT`$_1$** **T`-> ε** **F-> digit**	10	L3	CO2
	b.	Given D-> TL T-> int T-> float L-> L_1, id L-> id Give SDD for the above grammar and Develop a dependency graph for the declaration **float id_1, id_2, id_3**	10	L3	CO2
Module - 4					
Q7	a.	Apply the backpatching technique to translate boolean expressions into intermediate code using the following grammar: $B \rightarrow B_1 \|\| MB_2 \mid B_1 \&\& MB_2 \mid !B_1 \mid (B_1) \mid E_1$ rel $E_2 \mid$ true |false M→ ε Demonstrate the generation of intermediate code for x<100 || x>200 && x!=y.	10	L3	CO4
	b.	Construct a Syntax Directed Definition (SDD) to generate a Directed Acyclic Graph (DAG) for grammar E → E_1 + T E → E_1 - T E → T T → (E) T → id T → num Apply the SDD to construct the DAG for the expression: a + a * (b - c) + (b - c) * d	10	L3	CO4
OR					
Q8	a.	Apply the backpatching technique to translate flow-of-control statements into intermediate code. Construct the Syntax Directed Definition (SDD) and demonstrate the generation of three-address code for conditional and iterative statements such as **if, if–else**, and **while**. S-> if(B)S|if (B) S else S | while (B) S | { L}| A; L-> LS |S	10	L3	CO4
	b.	Construct the Three Address Code (TAC), Quadruples, Triples, and Indirect Triples for the expression: x := -a * b + -a * b.	10	L3	CO4
Module – 5					
Q9	a.	Use the instructions and addressing modes available in the target machine to generate target code for the statement: x = a + b * 5	10	L3	CO5

	b	Demonstrate the application of peephole optimization techniques such as redundant instruction elimination, algebraic simplification, and control flow optimization on a given instruction sequence.
		MOV R1, A
		ADD R1, 0
		MUL R2, 1
		MOV R3, R3
		SUB R4, 0
		JMP L1
		JMP L2
		L1: ADD R5, R6
			10	L3	CO5

		OR			
Q10	a.	Generate the code for the following sequence assuming that x,y, and z are in memory locations: if x < y goto L1 z = 0 goto L2 L1: z = 1 L2:	10	L3	CO5
	b.	Construct basic blocks and flow graph for the following code: 1. a = b + c 2. d = a - e 3. if d > 0 goto L1 4. a = d * e 5. goto L2 6. L1: a = b + c 7. L2: print a	10	L3	CO5

-*****-

|