VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

REG.NO.:

SLOT:A1+TA1

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - II
FALL SEMESTER 2025-2026

Programme Name & Branch : B.Tech CSE and Specialisation
Course Code and Course Name : BCSE307L - Compiler Design
Faculty Name(s) : Prof. MUTHUNAGAI S U, Prof. KRISHNARAJ N, Prof. NIVITHA K, Prof.ISLABUDEEN M, Prof. VISWANATHAN A, Prof. SUGANTHINI C, Prof. KANAGARAJ R, Prof. KATARI BALAKRISHNA, Prof. LAKSHMI S, Prof. BHAWANA TYAGI, Prof. BASKARAN P, Prof. SENDHIL KUMAR K.S, Prof. NAGA PRIYADARSINI R, Prof. VISHNU PRIYA A, Prof. BHUVANESWARI M
Class Number(s) : VL2025260101612, 1590, 1614, 1619, 1579, 1627, 1592, 1581, 1610, 1636, 1597, 1608, 1632, 1584, 1587
Date of Examination : 05.10.2025 (AN)
Exam Duration : 90 minutes **Maximum Marks: 50**

General instruction(s):

- Answer All Questions
- M - Max mark; CO – Course Outcome; BL – Blooms Taxonomy Level (1 – Remember, 2 – Understand, 3 – Apply, 4 – Analyse, 5 – Evaluate, 6 – Create)
- Course Outcomes (Type the CO statements covered in this question paper. Use the CO number as per the syllabus copy)
  CO2. Develop language specifications using context free grammars (CFG).
  CO3. Apply the ideas, the techniques, and the knowledge acquired for the purpose of developing software systems.
  CO4. Constructing symbol tables and generating intermediate code.
  CO5. Obtain insights on compiler optimization and code generation.

Q. No	Question	Modu	Marks	CO	BL
1.	Let the grammar G=(V,Σ,P,S) where: V = { Stmt, DeclStmt, Type, IdList, IdListTail }, Σ= { char, identifier, , , ; } and S = Stmt Production rules are defined as follows. a. A statement consists of a declaration statement. b. A declaration statement starts with a type, followed by a list of identifiers, and ends with a semicolon. c. Only the char type is supported in this definition. d. The identifier list begins with an identifier, followed by optional additional identifiers. e. Allows declaring multiple variables, separated by commas f. Ends the identifier list when there are no more variables. Formulate the grammar. Demonstrate the formulated grammar is LALR or not with a detailed justification. And consider any string w∈L(G) to illustrate the parsing process. G=(V,Σ,P,S) where: (2 Marks) V = { Stmt, DeclStmt, Type, IdList, IdListTail }, Σ= { char, identifier, , , ; } and S = Stmt	2	10	2	3

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - II
FALL SEMESTER 2025-2026

P:
1. Stmt → DeclStmt
2. DeclStmt → Type IdList ;
3. Type → char
4. IdList → identifier IdListTail
5. IdListTail → , identifier IdListTail
6. IdListTail → ε

Assumed : Stmt (S), DeclStmt (D), Type (T), IdList (I), char (c), identifier (i), IdListTail (L)

Representation of Canonical Collection of LR(1) Items (3 Marks)
Parsing Table (3 Marks)

I/S	c	i	;	,	$	S	D	T	I	L
I0	S3					1	2	4		
I1					Acc					
I2					R1					
I3		R3								
I4		S7							5	
I5			S6							
I6					R2					
I7			R6							8
I8			R4							
I9			S10							
I10				S7						11
I11			R4							

Justification: Yes, the grammar is LALR(1). No left recursion or common prefix ambiguity, No ambiguity or conflicts.
Parsing a w∈L(G) using stack, parsing table and input buffer (2 Marks)

2.	We have the following grammar with translation rules. Here S, T, and R are Non-terminals; id represents an integer; id.val gives its integer value; # and % are operators. $S \rightarrow S_1 \# T$ { _____ } $S \rightarrow T$ { S.val = T.val } $T \rightarrow T_1 \% R$ { _____ } $T \rightarrow R$ { T.val = R.val } $R \rightarrow id$ { R.val = id.val } When we evaluate an expression 20 # 10 % 5 # 8 % 2 % 2 using this translation scheme, the S.val is 80. Hence answer the following. (a) Fill the blanks with appropriate semantic rules. (2.5 Marks) $S \rightarrow S_1 \# T$ { $S.val = S_1.val * T.val$ } $T \rightarrow T_1 \% R$ { $T.val = T_1.val / R.val$ }	3	10	3	3

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - II
FALL SEMESTER 2025-2026

	(b) Write about operator precedence and associativity (2.5 Marks) High precedence operations %(division) Low precedence operations #(multiplication) The production rule $S \rightarrow S_1 \# T$ shows that to process a # operation, a T must be fully evaluated first. This structure is what explicitly establishes % as a higher-precedence operator than # within this particular grammar. Here both # and % are left associative operators. (c) Parse Tree Construction (2.5 Marks) (d) Bottom-up evaluation (2.5 Marks).				

3.	Represent the following code into Quadruple, triples and indirect triples form.
	#include <stdio.h>
	int main() {
	int i;
	int a[10];
	// Initialize array elements to 0 using for loop
	for (i = 0; i < 10; i++) {
	a[i] = 0;
	}
	return 0;
	}
	Three-Address Code:
	(1) i = 0
	(2) L1: if i >= 10 goto L2
	(3) a[i] = 0
	(4) i = i + 1
	(5) goto L1
	(6) L2:
	
	Quadruples
	Quadruples represent each operation as a 4-tuples (operator, arg1, arg2, result).
		4	10	4	2

Index	**Operator**	**Arg1**	**Arg 2**	**Result**
(1)	=	0		i
(2)	>=	i	10	
(3)	=	0		a[i]
(4)	+	i	1	i
(5)	goto			L1
(6)	label			L2

Triples

Triples represent each operation as a 3-tuples (operator, arg1, arg2) and use implicit result locations (indexes).

Index	Operator	Arg1	Arg2
(1)	=	0	
(2)	>=	i	10
(3)	=	0	
(4)	+	i	1
(5)	goto		(2)
(6)	Label		L2

Indirect Triple Format

This is a pointer table to the triples, allowing reordering without modifying the code directly.

Index	Points to Triple
(1)	(10)
(2)	(11)
(3)	(12)
(4)	(13)
(5)	(14)
(6)	(15)

Index	Operator	Arg1	Arg2
(1)	=	0	
(2)	>=	i	10
(3)	=	0	—
(4)	+	i	1
(5)	goto	—	(11)
(6)	Label		L2

(b). Write SDT to generate an intermediate code for the following code snippet.

```
while (a < b && c != d)
{
   count++;
}
count+=4;
```

Do you think back patching is required for the above case? Justify your answer with detailed explanation.

Yes. Backpatching is required. Backpatching is needed when the exact target labels of jumps are not known at the point where the jump instructions are generated. It allows filling in those jump targets later when they are known. (1 Mark)

	SDT to generate an intermediate code (2 Marks) Explanation about Backpatching process (2 Marks)				
4.	Answer the followings and design a syntax-directed definition for calculating the storage layout of a three-dimensional array declared with zero-based indexing. Consider the array declaration format: T A[d1][d2][d3]; Where: T is a primitive data type (int, float, or char), d1, d2, and d3 are the dimension sizes. (a) Complete the missing part of the grammar rule to represent the array declaration: (2 Marks) D → T ID [NUM1] [NUM2] [NUM3] ; T → int | float | char (b) Define the semantic rules required to compute the total storage size for the given array. (3 Marks) 1. D → T ID [NUM1] [NUM2] [NUM3] ; {D.d1 = NUM1.val, D.d2 = NUM2.val, D.d3 = NUM3.val, D.width = T.width, D.size = D.d1 × D.d2 × D.d3 × D.width} 2. T → int { T.width = 4} 3. T → float {T.width = 8} 4. T → char {T.width = 1} (c) Using a bottom-up parsing approach, demonstrate the step-by-step semantic evaluation and computation of the total size for a sample input declaration (e.g., int A[2][3][4];). (3 Marks) Parse Tree Construction, Semantic rules evaluation T.width=4, D.d1=2, D.d2=3, D.d3=4, D.width=4, D.size=2*3*4*4=96 (d) Specify the attributes and its types used in the syntax-directed definition. (2 Marks) Definition of Synthesized attributes and Inherited attributes. T.width, NUM1.val, NUM2.val, NUM3.val and D.size are Synthesized attributes.	4	10	4	3

5.	Translate the following code into three address code. Identify the basic blocks and construct a flow graph for the below code snippet.
	for (i = 2; i <= n; i++)
	{
	a[i] = TRUE;
	}
	count = 0;
	s = sqrt(n);
	for (i = 2; i <= s; i++) {
	if (a[i] == TRUE) {
	count++;
		5	10	5	2

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - II
FALL SEMESTER 2025-2026

---

	for (j = 2 * i; j <= n; j = j + i) {
	a[j] = FALSE;
	}
	}
	}
	Three address code (4 Marks)
	Basic Blocks (3 Marks)
	Flow Graph (3 Marks)				

****