VIT
Vellore Institute of Technology

School of Computer Science and Engineering

Fall Semester 2023-24 UG Seniors (B.Tech 2021 Batch)

Continuous Assessment Test – I

SLOT: C1+TC1, C2 +TC2

Programme Name & Branch : B.Tech - All Core CS and its Specialization branches

Course Name & code: Compiler Design - BCSE307L

Exam Duration: 90 Min. **Maximum Marks: 50**

General instruction(s):

Answer ALL Questions

Q.No.	Question	Max. Marks
1.	a) Look at the following instruction: total_marks=60% of internal_marks + 40% of external_marks Feed the above instruction as input in the way that your compiler understands and show the result of transformation of your input in each stage of the compiler and find the target assembly-level code from it. (7 Marks) b) Find the tokens and total token count in the following instruction. **printf("%d Hai", &x);** (3 Marks)	10
2.	Using Direct method, convert the following Regular Expression into Deterministic Finite State Automata. $(a^{+} \mid \varepsilon \mid b^{+}).c$ (Note: ε denotes Epsilon)	10
3.	a) Check whether the following grammar is ambiguous or not. If it's ambiguous remove the ambiguity from it. And cross check the string **"if con then if con then other else other"** is derivable through only one parse tree from your unambiguous grammar. **stmt →if con then stmt | if con then stmt else stmt | other** (5 Marks)	10

	b) In the pursuit of obtaining unambiguous grammar, a student (Say X) applied disambiguity rules and finally got the resultant grammar shown below: **E→ F ¥ E | F $ E | F** **F→ F § E | constant** What is your inference about precedence and associativity of the operators ¥ ,$ and § present in the above grammar. Also, Calculate this expression: **3 § 5 ¥ 7 $ 9 § 5** (Note: ¥ performs multiplication, $ performs addition and § performs subtraction.) (5 Marks)	
4.	Construct the recursive descent parser (with no backtracking) for the following grammar. Write your recursive procedure in a clear manner with justification as comments. Also parse the string **"5+3*7"** through your recursive procedure. **Expression→Expression+Term | Term** **Term →Term*Factor | Factor** **Factor → constant | (Expression)**	10
5.	Check whether the following grammar is LL (1) or not through constructing its parse table. If it is LL(1), parse the string **"cccd"** S→ A A→Bb | Cd ***B → aB | ε*** C → cC | ε	10