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 – I Answer key
FALL SEMESTER 2025-2026

Programme Name & Branch : B.Tech CSE with All Specialization
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

ClassNumber(s): VL2025260101612,1590,1614,1619,1579,1627,1592,1581,1610,1636, 1597,1608,1632,1584,1587
Date of Examination : 17.08.2025 FN
Exam Duration : 90 minutes **Maximum Marks: 50**

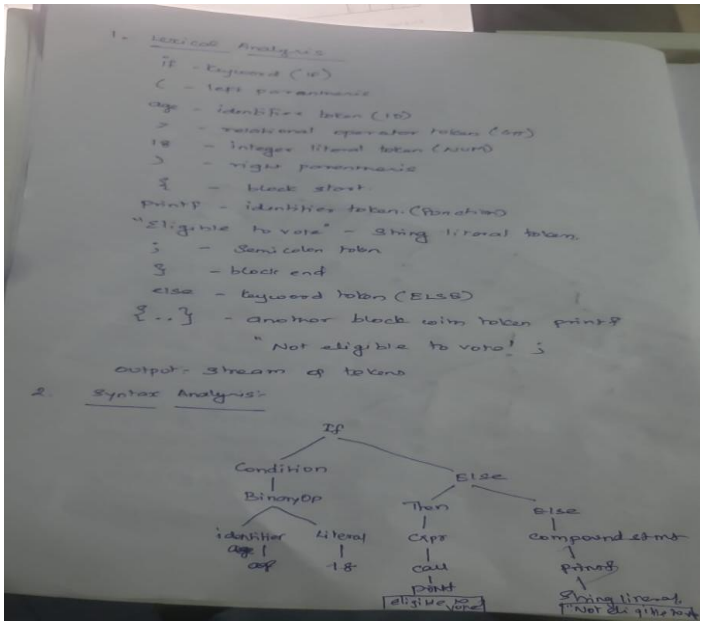
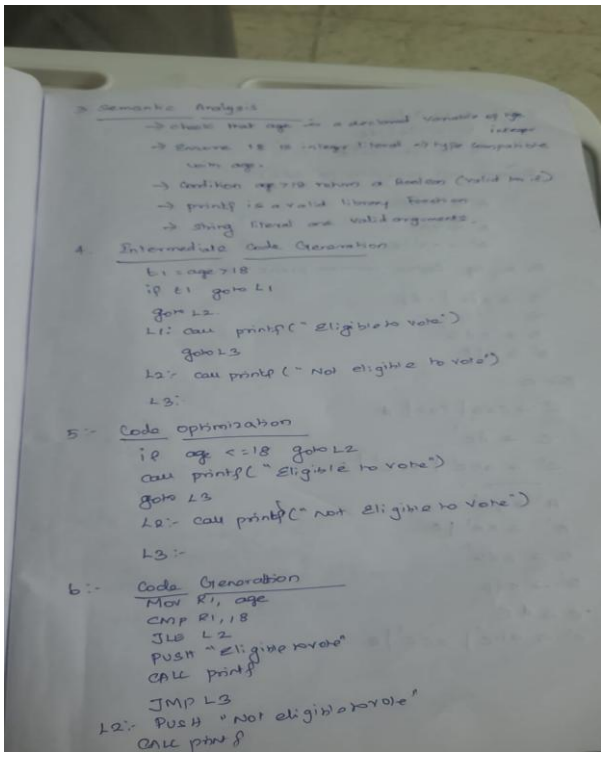
General instruction(s):

- 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:
  CO1: Apply the skills on devising, selecting, and using tools and techniques towards compiler design
  CO2. Develop language specifications using context free grammars (CFG).

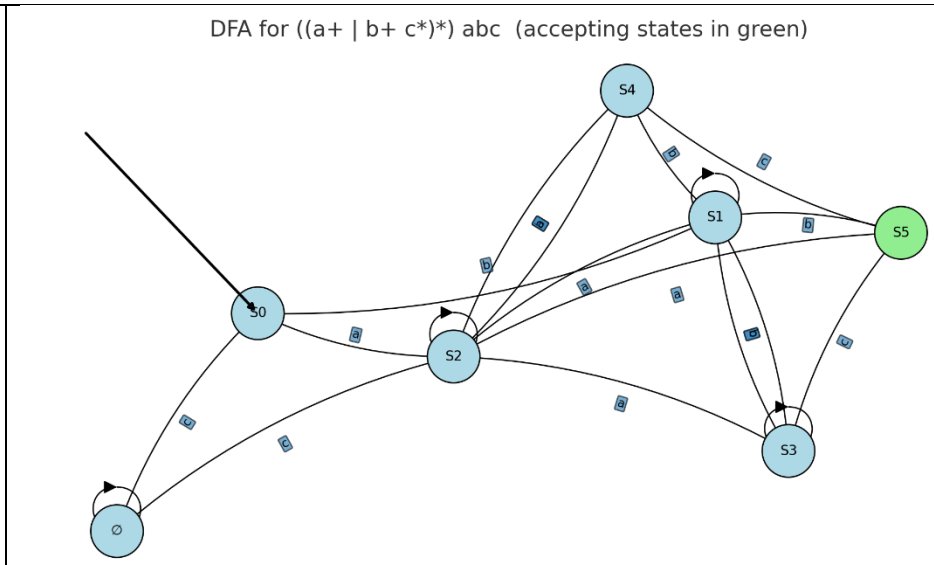
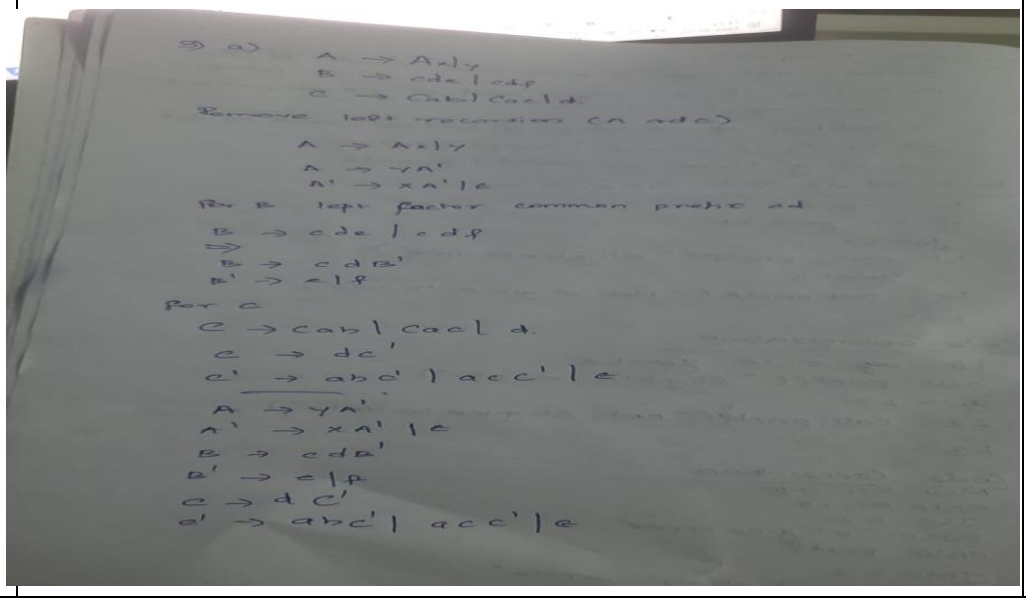
Q. No	Question	M	CO	BL

1.	Explain the phases of compiler in detail and analyse how the following code snippet is processed in each phase:
	if(age>18)
	{
	printf(“Eligible to vote”);
	}
	else
	{
	printf(“Not eligible to vote”);
	}
	Answer:
		10	1	3

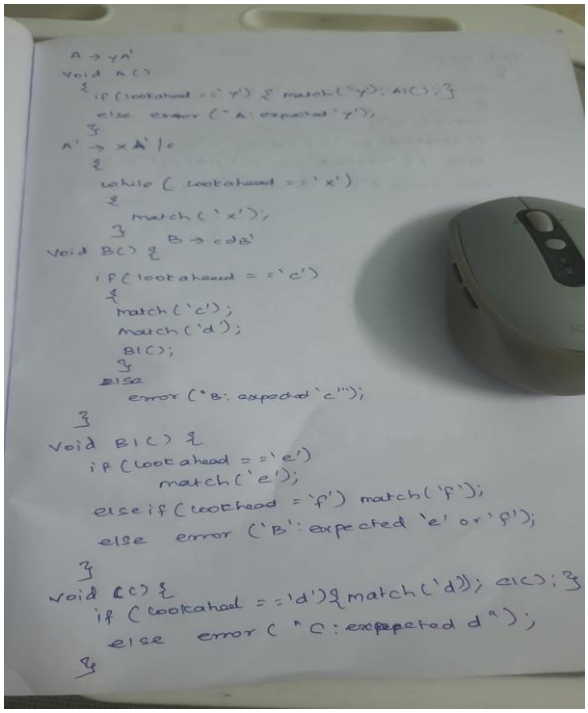
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST – I Answer key
FALL SEMESTER 2025-2026

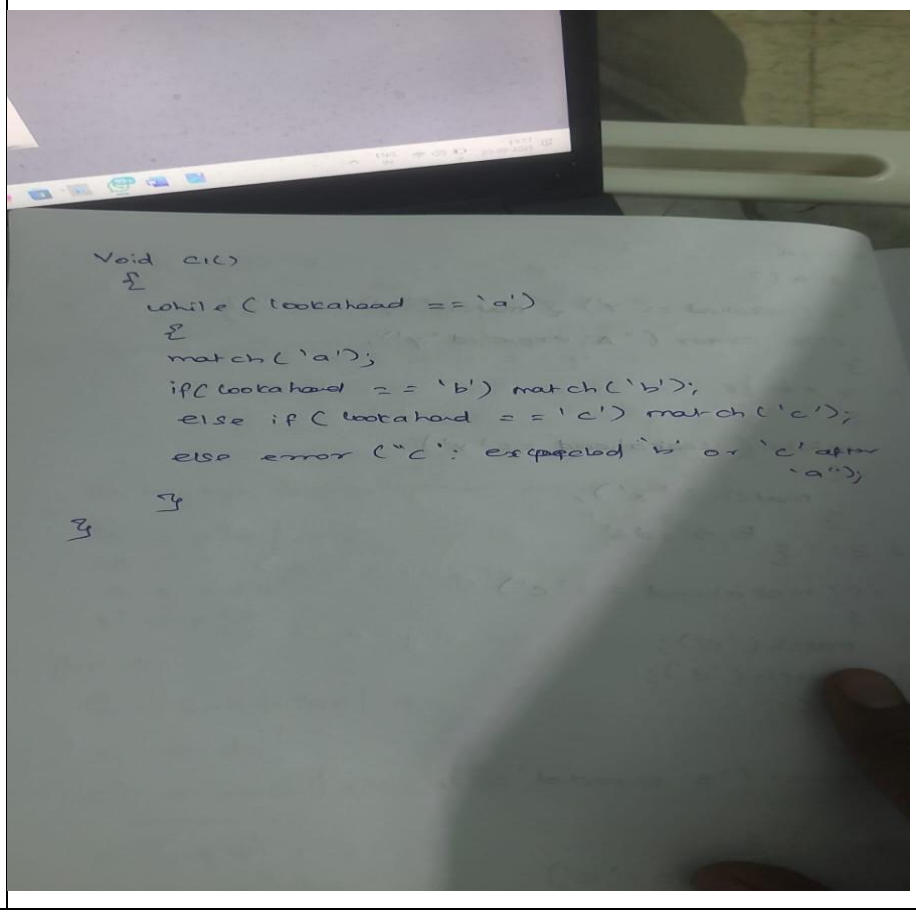
	 			
2.	Construct DFA for the regular Expression **$(a^+\|b^+c^*)^*\varepsilon\ (abc)$** using direct method with all the necessary steps Answer:	10	1	3

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST – I Answer key
FALL SEMESTER 2025-2026

				
3.	a) Construct top-down parser using recursive approach for the following grammar with all the necessary steps A → Ax |y B → cde | cdf C → Cab| Cac | d The starting Symbol for the grammar is 'A' Answer: 	7	2	3

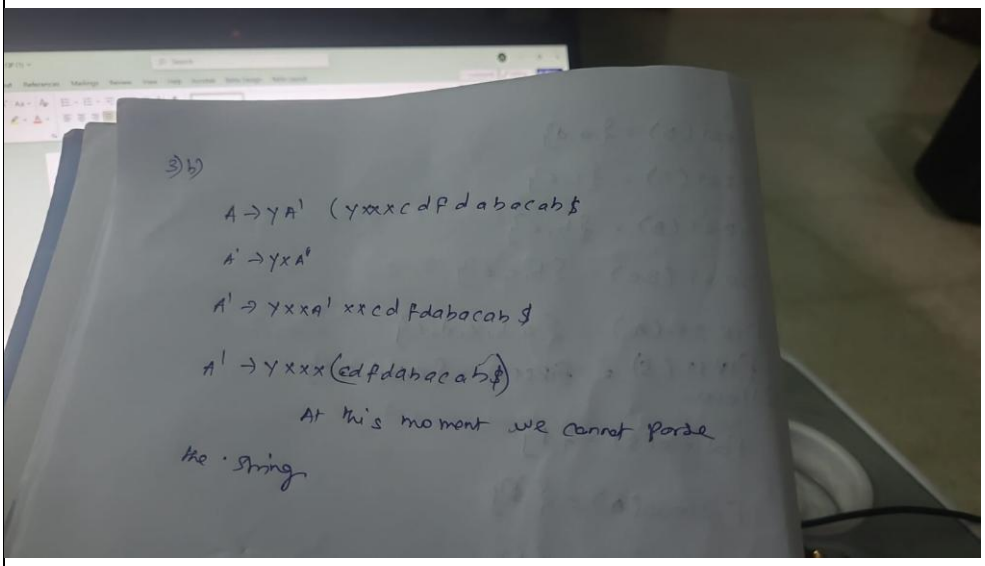
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST – I Answer key
FALL SEMESTER 2025-2026



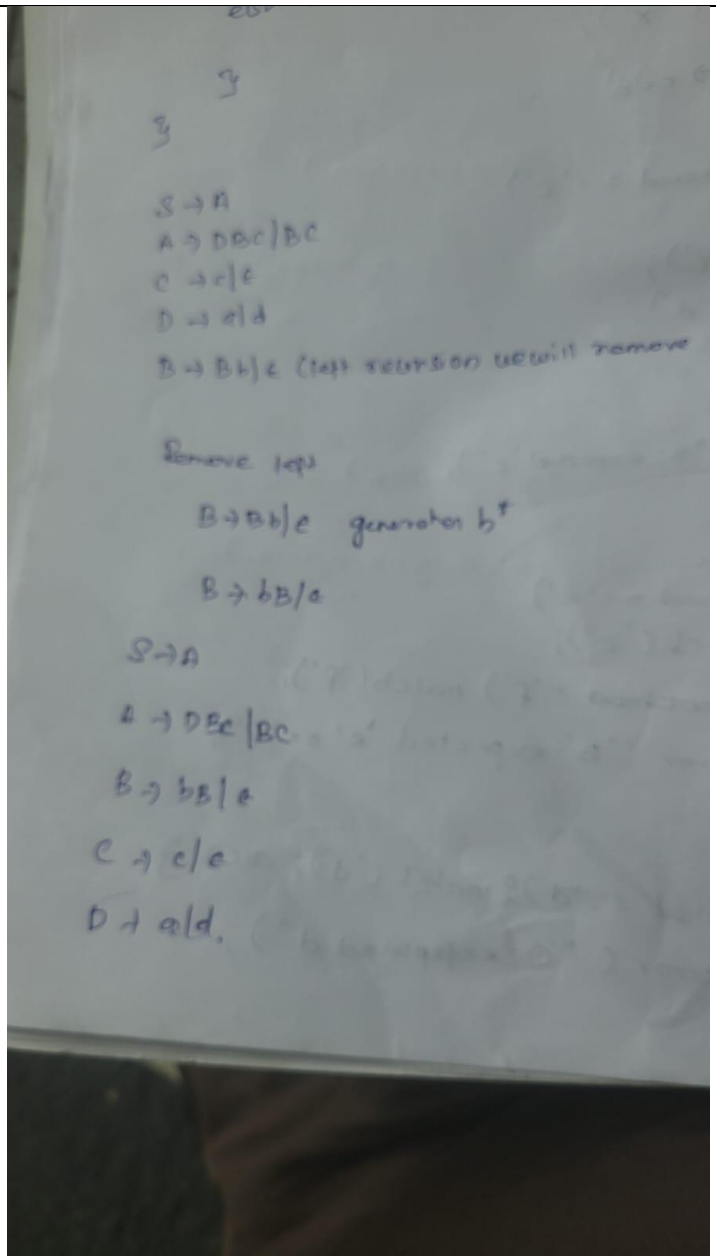


SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST – I Answer key
FALL SEMESTER 2025-2026

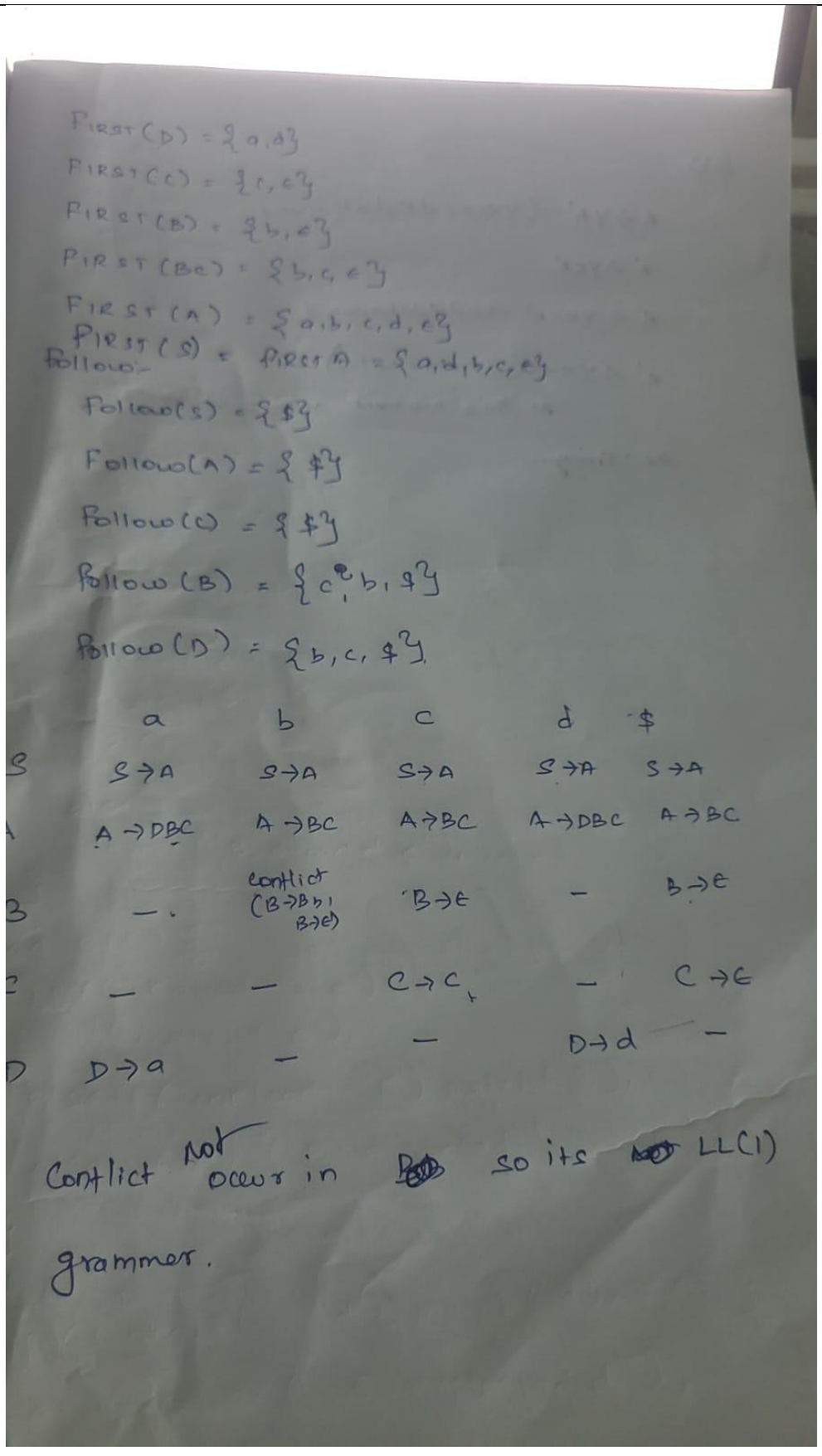
---

	b) Check whether the given string is parsed by using above grammar (refer 3a)) or not "yxxxcdfdabacab" with the steps Answer:  	3		
4.	Consider the Grammar G as follows: S → A A → DBC / BC C → c / ε D → a / d B → Bb / ε Apply LL(1) parsing and construct the parsing table for the above grammar and justify it whether it is LL(1) or not using string "abbc" and the starting State for the above grammar is "S" Answer:	10	2	3

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST – I Answer key
FALL SEMESTER 2025-2026



SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST – I Answer key
FALL SEMESTER 2025-2026



FIRST(D) = {a, ∈}
FIRST(C) = {c, ∈}
FIRST(B) = {b, ∈}
FIRST(Bc) = {b, c, ∈}
FIRST(A) = {a, b, c, d, ∈}
FIRST(S) = FIRST(A) = {a, d, b, c, ∈}

Follow-
Follow(S) = {$}
Follow(A) = {$}
Follow(C) = {$}
Follow(B) = {c, b, $}
Follow(D) = {b, c, $}

	a	b	c	d	$
S	S→A	S→A	S→A	S→A	S→A
A	A→DBC	A→BC	A→BC	A→DBC	A→BC
B	—	conflict (B→Bb | B→∈)	B→∈	—	B→∈
C	—	—	C→c	—	C→∈
D	D→a	—	—	D→d	—

Conflict not occur in so its LL(1) grammer.

	. Start stack: S $. Show step-by-step: (Stack top on left; Input pointer shows remaining input) Initial: Stack S $, Input a b b c $ Action: table[S, a] = S → A. Replace S by A. Step 1: Stack A $, Input a b b c $ Action: table[A, a] = A → D B C. Replace A by D B C (push in right-to-left: C then B then D). Step 2: Stack D B C $, Input a b b c $ Top = D. table[D, a] = D → a. So match D→a: pop D and match input a. After matching: Input becomes b b c $. Stack now B C $. Step 3: Stack B C $, Input b b c $ Top = B. table[B, b] = B → b B. So expand B → b B. Replace B by b B. Step 4: Stack b B C $, Input b b c $ Top = terminal b. Match terminal b with input b: pop b and advance input. After matching: Input b c $. Stack B C $. Step 5: Stack B C $, Input b c $ Top = B. table[B, b] = B → b B. Expand again.			

	Step 6: Stack b B C $, Input b c $ Top b (terminal) — match input b. After match: Input c $. Stack B C $. Step 7: Stack B C $, Input c $ Top = B. table[B, c] = B → ε. So pop B (apply ε, do nothing to input). Stack now C $. Step 8: Stack C $, Input c $ Top = C. table[C, c] = C → c. Expand and match: pop C, push c. Then match terminal c with input c. After match: Input $. Stack $. Step 9: Stack $, Input $ Both are $ → accept.			
5.	Construct an Operator Precedence Parser for the following grammar. Also, build the operator precedence relation table and the operator precedence function table. Show the parsing steps for the input string: "i - i / i" Expr → Expr-Expr Expr → Expr / Expr Expr → (Expr) Expr → i Answer :	10	2	3

Terminals order: `i, -, /, (, ), $`

`<` means "yields precedence" (shift), `=` means "equal", `>` means "takes precedence" (reduce). Blank cells are errors / not applicable.

from \ to	i	-	/	(	)	$
i	>	>	>		>	>
-	<	>	<	<	>	>
/	<	>	>	<	>	>
(	<	<	<	<	=	
)	>	>	>		>	>
$	<	<	<	<		=

One valid choice (many choices possible) that realizes the relation table above is:

terminal	i	-	/	(	)	$
f	8	3	5	0	9	0
g	6	2	4	7	0	0

Check a couple entries: $f(i) = 8 > g(-) = 2 \Rightarrow i > -$; $f(-) = 3 < g(/) = 4 \Rightarrow - < /$; $f('(') = 0 = g(')') = 0 \Rightarrow (=)$. This numeric table is just one convenient realization.

Parsing the input i - i / i

Input with end marker: i - i / i $

We will show the canonical operator-precedence parse trace. At each step show the stack (bottom on left), the remaining input (next symbol underlined), and the action (shift or reduce and what was reduced).

Start: Stack = $, Input = i - i / i $

(we compare the nearest terminal on the stack with the next input terminal using the precedence table above)

Stack: $
Input: i - i / i $
Compare $ < i → shift i.
Stack → $ i

Stack: $ i
Input: - i / i $

	Compare i > - → reduce the handle i → reduce i to E. Stack → $ E (in terminal-scan we still consider nearest terminal to left = $) Stack: $ E Input: - i / i $ Nearest terminal on stack is $. Compare $ < - → shift -. Stack → $ E - Stack: $ E - Input: i / i $ Compare - < i → shift i. Stack → $ E - i Stack: $ E - i Input: / i $ Compare i > / → reduce i → E. Stack → $ E - E Stack: $ E - E Input: / i $ Nearest terminal on stack is -. Compare - < / → shift /. Stack → $ E - E / Stack: $ E - E / Input: i $ Compare / < i → shift i. Stack → $ E - E / i Stack: $ E - E / i Input: $ Compare i > $ → reduce i to E. Stack → $ E - E / E Stack: $ E - E / E Input: $ Nearest terminal is /. Compare / > $ → reduce the rightmost handle E / E → reduce to E. Stack → $ E - E			

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST – I Answer key
FALL SEMESTER 2025-2026

	Stack: $ E - E Input: $ Nearest terminal is -. Compare - > $ → reduce E - E → reduce to E. Stack → $ E Stack: $ E Input: $ Compare $ = $ → accept. Interpretation: the reductions happened in such a way that E / E was reduced before E - E — i.e. the input i - i / i was parsed as			

****