S.No.:01_JP_CS_11072025

MADE EASY
India's Best Institute for IES, GATE & PSUs

Delhi | Bhopal | Hyderabad | Jaipur | Pune | Bhubaneswar | Kolkata

Web: www.madeeasy.in | **E-mail:** info@madeeasy.in | **Ph:** 011-45124612

COMPLIER DESIGN

COMPUTER SCIENCE & IT

Date of Test : 11/07/2025

ANSWER KEY ➤

1.	(c)	7.	(a)	13.	(c)	19.	(a)	25.	(c)
2.	(d)	8.	(c)	14.	(d)	20.	(a)	26.	(c)
3.	(b)	9.	(a)	15.	(d)	21.	(a)	27.	(d)
4.	(b)	10.	(d)	16.	(b)	22.	(d)	28.	(b)
5.	(c)	11.	(b)	17	(d)	23.	(c)	29.	(d)
6.	(d)	12	(c)	18.	(c)	24.	(d)	30.	(d)

DETAILED EXPLANATIONS

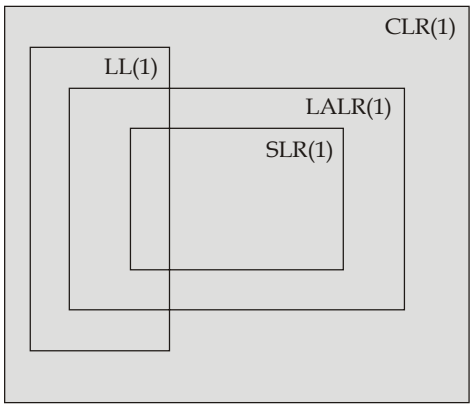
1. (c)

Simple two-pass assembler:

1. Allocates space for the literals.
2. Computers the total length of program (syntax analysis).
3. Builds the symbol table for the symbols and their values.

2. (d)

Relation between LL(1), SLR(1) and CLR(1) and LALR(1) given below:



S_1 is false, S_2 is true and S_3 is false.

3. (b)

In both stack and heap allocation, memory allocated at runtime.

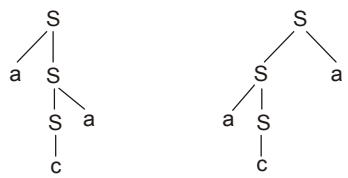
Static allocation does not support recursion. However, in stack allocation, storage is organized as a stack and activation records are pushed and popped as activation begin and end respectively.

4. (b)

- A token is a pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit e.g. a particular keyword or a sequence of input characters denoting an identifier.
- A pattern is a description of the form that the lexemes of a token may take. In the case of a keyword as a token, the pattern is just the sequence of characters that form the keyword.
- A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

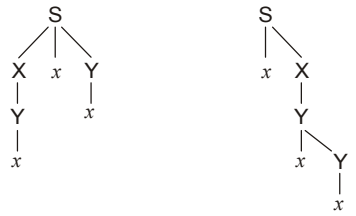
5. (c)

G_1 : Let '*w*' be '*aca*'



Since two parse trees are possible for a string, hence the Grammar is ambiguous.

G_2 : Let '*w*' be '*xxx*'

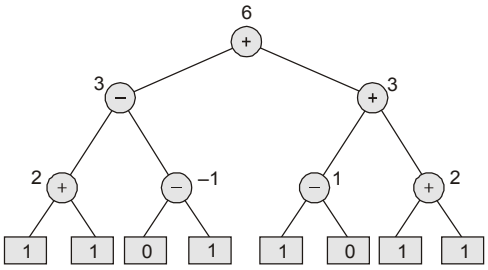


Since two parse trees are possible for a string, hence the Grammar is ambiguous.
So both grammars are ambiguous.

6. (d)

$$\begin{aligned} \text{FIRST } \{B\} &= \{b, \in\} \\ \text{FOLLOW } \{B\} &= (\text{FIRST } (C) - \{\in\}) \cup \text{FOLLOW } (A) \cup \text{FIRST } (b) \\ &= \{c\} \cup \{\$\} \cup \{b\} \\ &= \{b,c, \$\} \end{aligned}$$

7. (a)

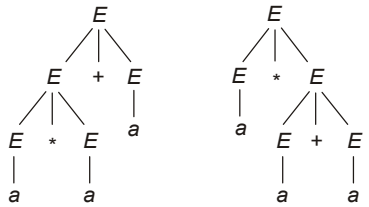


8. (c)

x is inherited.
y is synthesized.

9. (a)

$a * a + a$



Since, two parse trees are possible.
Hence, grammar is ambiguous.

10. (d)

Left recursion is there in the production Rule 2 and 3.

Rule 2 :

$A \rightarrow Aba \mid b$

After removing left recursion :

$A \rightarrow bA'$

$A' \rightarrow baA' \mid \in$

Rule 3 :

$B \rightarrow BaA \mid a$

After removing left recursion :

$B \rightarrow aB'$

$B' \rightarrow aAB' \mid \in$

Hence, the grammar after resolving left recursion is,

$S \rightarrow AB \mid BA$

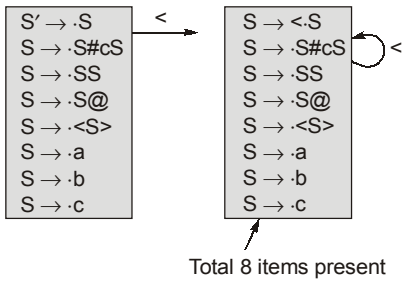
$A \rightarrow bA'$

$A' \rightarrow baA' \mid \in$

$B \rightarrow aB'$

$B' \rightarrow aAB' \mid \in$

11. (b)



12 (c)

Stack contains only a set of viable prefixes.

13. (c)

$$\text{Follow (S)} = \text{First (L')}$$
$$\text{First (L')} = \{*, \in\}$$

Since, First (L') contain $\in$, hence

$$\text{Follow (S)} = \{\text{First (L')} - \in\} \cup \text{follow (L)}$$
$$\text{Follow (L)} = \{), \{\text{follow (S)}\}\}$$
$$\text{Follow (S)} = \$$$
$$\text{Follow (L)} = \{), \$\}$$
$$\text{Follow (S)} = \{*,), \$\}$$

Hence, there are three elements in the follow set of Non-terminal S.

14. (d)

```
int main ( )
{
    int a, b;          // initialize integer a, b
    a = 10;
    b = 15;
    Printf("a = %d, b = %d", a++ , b--);
}
```

Total 30 tokens are available in the above C program.

15. (d)

$3 - 2 * 4 \$ 2 \$ 3$

$= (((3 - 2) * 4) \$ 2) \$ 3$

$= ((1 * 4) \$ 2) \$ 3$

$= (4 \$ 2) \$ 3$

$= 16^3 = 4096$

16. (b)

Checkit out using following code.

MOV	a, R_1	
opr	b, R_1	$t_1 = a + b$
MOV	d, R_2	
opr	c, R_2	$t_2 = c + d$
opr	e, R_2	$t_3 = e - t_2$
MOV	t_3, R_1	
opr	t_1, R_1	$t_4 = t_1 - t_3$

Minimum number of MOV instructions required = 3.

17 (d)

$$IN = USE \cup \{OUT - DEF\}$$

$$OUT = \cup IN \text{ (successor)}$$

			FIRST GO		SECOND GO		THIRD GO	
Block	**USE**	**DEF**	**IN**	**OUT**	**IN**	**OUT**	**IN**	**OUT**
B1	{m, n}	{a, i, j}	{m, n}	{i, j}	{m, n}	{a, i, j}	{m, n}	{a, i, j}
B2	{i, j}	{i, j}	{i, j}	{a}	{a, i, j}	{a, j}	{a, i, j}	{a, j}
B3	ϕ	{a}	ϕ	{a}	ϕ	{a, j}	{j}	{a, j}
B4	{a}	{i}	{a}	{i, j}	{a, j}	{a, i, j}	{a, j}	{a, i, j}

∴ The variables that are live at exit (i.e. live out) of each basic block are

B1 = {a, i, j}, B2 = {a, j}

B3 = {a, j}, B4 = {a, i, j}

18. (c)

	FIRST	FOLLOW
S	{a, b, ∈}	{a, b, $}
A	{a, b, ∈}	{a, b}
B	{a, b, ∈}	{a, b, $}

LL(1) Parsing table:

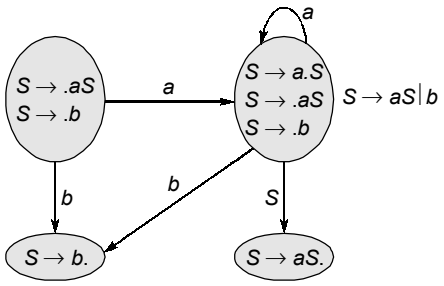
	a	b	$
S	$S \to aAbB$ $S \to \in$	$S \to bAbB$ $S \to \in$	$S \to \in$
A	$A \to S$	$A \to S$	
B	$B \to S$	$B \to S$	$B \to S$

There are only 2 entries in which there are multiple productions.

19. (a)

- Statement I and IV is correct.
- Type checking is done at semantic analysis phase.
- Target code generation is dependent based on the machine.
- Symbol table is accessed during lexical, syntax and semantic analysis phase.

20. (a)

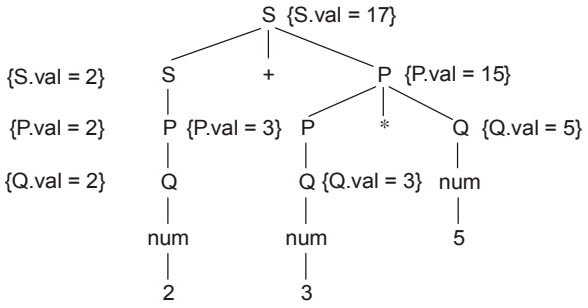


Some possible stack contents are

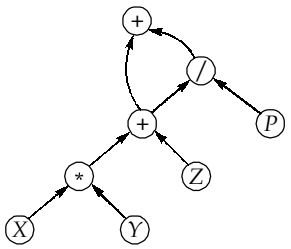
aaS, *ab*, *b*, etc.

21. (a)

Given SDT is S-attributed and hence L-attributed too. Since all translations are appended at end and attributes are synthesis, hence both L-attributed and S-attributed approach evaluates to same value.



22. (d)



Total 8 edges

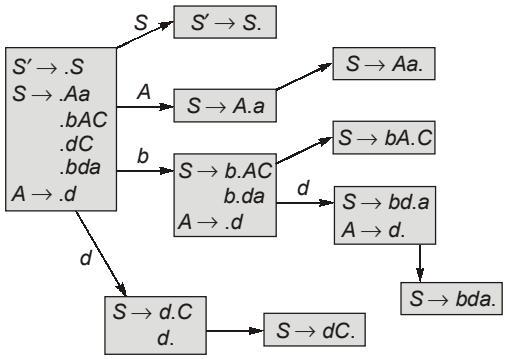
23. **(c)**

$$\begin{aligned} t_1 &= a * b \\ t_2 &= -t_1 \\ t_3 &= c + d \\ t_4 &= -(a * b) + (c + d) \\ t_1 &= a + b \\ t_2 &= t_1 + t_3 \\ t_5 &= -(a * b) + (c + d) - (a + b + c + d) \end{aligned}$$

24. **(d)**

- Statement S_1 and S_2 are correct.
- Statement S_3 is incorrect. Heap and stack both are present in main memory.

25. **(c)**



Given grammar is not SLR (1), but LAR (1).

26. **(c)**

Static scoping means that x refers to the x declared innermost scope of declaration. Since 'h' is declared inside the global scope, the innermost x is the one in the global scope (it has no access to the x's in 'f' and 'g', since it was not declared inside them), so the program prints 23 twice.

Dynamic scoping means that x refers to the x declared in the most recent frame of the call stack. 'h' will use the x from either 'f' or 'g', whichever one that called it so the program would print 22 and 45.