①

PART A

1. c
2. b
3. c
4. b
5. a
6. b
7. b
8. c
9. d
10. c

PART B

11. $E \rightarrow E+T \mid E-T \mid T$
$T \rightarrow a \mid b \mid (E)$

Left Recursion

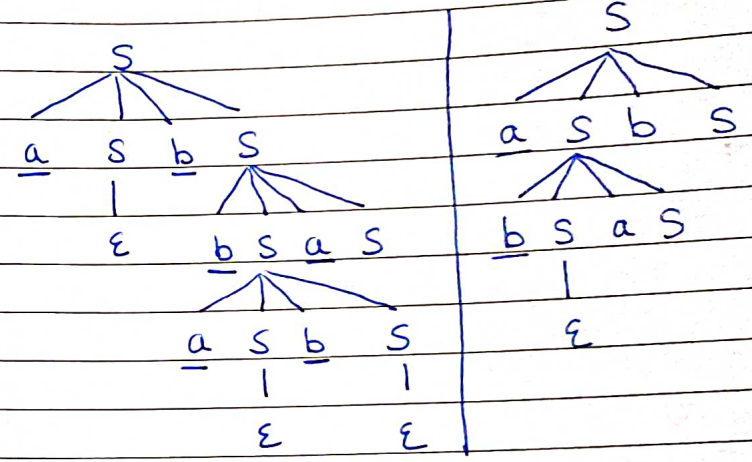
$E \rightarrow TE'$
$E' \rightarrow +TE' \mid -TE' \mid \varepsilon$
$T \rightarrow a \mid b \mid (E)$

Left Factoring

$E \rightarrow EE' \mid T$
$E' \rightarrow +T \mid -T \mid \varepsilon$
$T \rightarrow a \mid b \mid (E)$

12. $S \rightarrow aSbS \mid bSaS \mid \varepsilon$

$\omega = abbaba$



As we not able to draw two parse trees for the given string, the grammar is not ambiguous.

(2)

13. RESOURCE → RESOURCE - TEACHER | TEACHER
TEACHER → TEACHER / FACULTY | TEACHER
FACULTY → FACULTY * PROFESSOR | PROFESSOR
PROFESSOR → (RESOURCE) | job | role

Leading (RESOURCE) = { Leading (RESOURCE), - , Leading (TEACHER) }
= { - , / }

Leading (TEACHER) = { Leading (TEACHER), / }
= { / }

Leading (FACULTY) = { Leading (FACULTY), * , Leading (PROFESSOR) }
= { *, (, job, role }

Leading (PROFESSOR) = { (, job, role }

Trailing (RESOURCE) = { Trailing (TEACHER), - }
= { *,), job, role, /, - }

Trailing (TEACHER) = { Trailing (FACULTY), / , Trailing (TEACHER) }
= { *,), job, role, / }

Trailing (FACULTY) = { Trailing (PROFESSOR), * }
= { *,), job, role }

Trailing (PROFESSOR) = {), job, role }

14. Input string = abbcde

S → aABe
A → Abc | b
B → d

Stack	Input	Action
$	abbcde$	shift
$a	bbcde$	shift
$ab	bcde$	Reduce A → b
$aA	bcde$	shift
$aAb	cde$	shift
$aAbc	de$	Reduce A → Abc
$aA	de$	shift
$aAd	e$	Reduce B → d
$aAB	e$	shift
$aABe	$	Reduce S → aABe
$S		Accept

The string abbcde is accepted.

15. Compare and Contrast SLR, CLR and LALR

- LALR (Look Ahead - LR) is often used in practice, because the table obtained by it are considerably smaller than the canonical LR tables, yet most common syntactic constructs of programming languages can be expressed conveniently by an LALR grammar.
- For a comparison of parser size, the SLR and LALR tables for a grammar always have the same number of states, and this number is typically several hundred states for a language like C.
- The CLR table would typically have several thousand states for the same size language.
- It is much easier and more economical to construct SLR and LALR tables than the CLR tables.

PART - C

16. Predictive Parsing Table.

$S \rightarrow a \mid \wedge \mid (R)$

$T \rightarrow S, T \mid S$

$R \rightarrow T$

Step1 - Eliminate left Recursion

No left Recursion

Step2 - Eliminate left factoring

No left factoring

Step3 - Find First & Follow

FIRST (S) = {a, ∧, (}	FOLLOW (S) = {$, ,,)}
FIRST (T) = {a, ∧, (}	FOLLOW (T) = {)}
FIRST (R) = {a, ∧, (}	FOLLOW (R) = {)}

Step4 - Construct Predictive Parsing Table

		a	∧	(	)	,	$
S → a	S	S → a	S → ∧	S → (R)			
S → ∧	T	T → S, T T → S	T → S, T T → S	T → S, T T → S			
S → (R)	R	R → T	R → T	R → T			
T → S, T							
T → S							
R → T							

String, ω = (a, (a, a))

Stack	Input	Output
$ S	(a, (a,a)) $	
$) R (	(a, (a,a)) $	S → (R)
$) R	a, (a,a)) $	
$) T	a, (a,a)) $	R → T
$) T, S	a, (a,a)) $	T → S, T
$) T, a	a, (a,a)) $	S → a
$) T,	, (a,a)) $	
$) T	(a,a)) $	
$) S	(a,a)) $	T → S
$)) R (	(a,a)) $	S → (R)
$)) R	a,a)) $	
$)) T	a,a)) $	R → T
$)) T, S	a,a)) $	T → S, T
$)) T, a	a,a)) $	S → a
$)) T,	,a)) $	
$)) T	a)) $	
$)) S	a)) $	T → S
$)) a	a)) $	S → a
$))	)) $	
$)	) $	
$	$	Accept

18. LALR Parsing Table

S → CC

C → cC | d

Step 1 - Find First for all non Terminals.

FIRST(S) = {c, d}

FIRST(C) = {c, d}

Step 2 - Number all the productions

1. S → CC
2. C → cC
3. C → d

Step 3 - Write the Augmented Grammar

S' → S

S → CC

C → cC

C → d

Step 4 - Construct the canonical collection of sets of LR(1) items

closure of {[S' → • S, $]}

I0:
- S' → • S, $
- S → • CC, $
- C → • cC, c|d
- C → • d, c|d

goto (I0, S) - Closure of {[S' → S•, $]}

S' → S•, $ - I1

goto (I_0, C) - closure of $\{[S \to C.C, \$]\}$

$S \to C.C, \$$
$C \to .cC, \$$
$C \to .d, \$$
$\}\ I_2$

goto (I_0, c) - closure of $\{[C \to c.C, c|d]\}$

$C \to c.C, c|d$
$C \to .cC, c|d$
$C \to .d, c|d$
$\}\ I_3$

goto (I_0, d) - closure of $\{[C \to d., c|d]\}$

$C \to d., c|d$ $\}\ I_4$

goto (I_2, C) - closure of $\{[CC., \$]\}$

$S \to CC., \$$ $\}\ I_5$

goto (I_2, c) - closure of $\{[C \to c.C, \$]\}$

$C \to c.C, \$$
$C \to .cC, \$$
$C \to .d, \$$
$\}\ I_6$

goto (I_2, d) - closure of $\{[C \to d., \$]\}$

$C \to d., \$$ $\}\ I_7$

goto (I_3, C) - closure of $\{[C \to cC.c|d]\}$

$C \to cC., c|d$ $\}\ I_8$

goto (I_3, c) - closure of $\{[C \to c.C, c|d]\}$

$C \to c.C, c|d$
$C \to .cC, c|d$
$C \to .d, c|d$
$\}\ I_3$

goto (I_3, d) - closure of $\{[C \to d., c|d]\}$

$C \to d., c|d$ $\}\ I_4$

goto (I_6, C) - closure of $\{[C \to cC., \$]\}$

$C \to cC., \$$ $\}\ I_9$

goto (I_6, c) - closure of $\{[C \to c.C, \$]\}$

$C \to c.C, \$$
$C \to .cC, \$$
$C \to .d, \$$
$\}\ I_6$

goto (I_6, d) - closure of $\{[C \to d., \$]\}$

$C \to d., \$$ $\}\ I_7$

Steps - Construct the 2A2R(1) collection of items.

I_0:
$S' \to .S, \$$
$S \to .CC, \$$
$C \to .cC, c|d$
$C \to .d, c|d$

I_1:
$S' \to S., \$$

I_2:
$S \to C.C, \$$
$C \to .cC, \$$
$C \to .d, \$$

I_{36}:
$C \to c.C, c|d|\$$
$C \to .cC, c|d|\$$
$C \to .d, c|d|\$$

I_{47}:
$C \to d., c|d|\$$

I_5:
$S \to CC., \$$

I_{89}:
$C \to cC., c|d|\$$

(6)

LALR Parsing Table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	S36	S47		1	2
1			acc		
2	S36	S47			5
36	S36	S47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

19. $S \rightarrow S+S \mid S*S \mid id$

Leading(S) = {+, *, id}

Trailing(S) = {*, +, id}

$ <· +	+ ·> $
$ <· *	* ·> $
$ <· id	id > $

$S \rightarrow S+S$	$S \rightarrow S*S$
+ <· +	* <· +
+ <· *	* <· *
+ <· id	* <· id
+ ·> +	* ·> +
+ ·> *	* ·> *
id ·> +	id ·> *

Operator Precedence Relation Table.

	+	*	id	$
+	<·, ·>	<·, ·>	<·	·>
*	<·, ·>	<·, ·>	<·	·>
id	·>	·>		·>
$	<·	<·	<·	·>

Parsing the string

STACK	INPUT	ACTION
$	id+id*id$	shift
$id	+id*id$	Pop
$	+id*id$	shift
$+	id*id$	shift
$+id	*id$	Pop
$+	*id$	pop
$	*id$	shift
$*	id$	shift
$*id	$	Pop
$*	$	Pop
$	$	Accept

17,

$S \rightarrow int\ z\ x$
$Z \rightarrow a \mid b \mid -$
$X \rightarrow Y\ X \mid \varepsilon$
$Y \rightarrow a \mid b \mid 1 \mid 2$

FIRST (S) = {int}	FOLLOW (S) = {$}
FIRST (Z) = {a, b, -}	FOLLOW (Z) = {a, b, 1, 2, $}
FIRST (X) = {a, b, 1, 2, ε}	FOLLOW (X) = {$}
FIRST (Y) = {a, b, 1, 2}	FOLLOW (Y) = {a, b, 1, 2, $}

Non Recursive Pridictive Table

	int	a	b	1	2	-	$
S	S→int z x						
Z		Z→a	Z→b			Z→-	
X		X→Y X	X→Y X	X→Y X	X→Y X		X→ε
Y		Y→a	Y→b	Y→1	Y→2		

Stack	Input	Action
S $	int - a 1 $	
int z x $	int - a 1 $	S → int z x
z x $	- a 1 $	
- x $	- a 1 $	Z → -
x $	a 1 $	
Y X $	a 1 $	X → Y X
a x $	a 1 $	Y → a
x $	1 $	
Y X $	1 $	X → Y X
1 X $	1 $	Y → 1
X $	$	
$	$	X → ε

The string is Accepted.