Compiler Design Assignment - 1

LL(1) parsing table:

Given Grammar:

S -> W

W -> ZXY | XY

Y -> c | epsilon

Z -> a | d

X -> Xb | epsilon

Step 1: Compute FIRST and FOLLOW sets

FIRST(Y) = {c, epsilon}

FIRST(Z) = {a, d}

FIRST(X) = {b, epsilon}

FIRST(W) = FIRST(ZXY) U FIRST(XY)

    = {a, d} U {b, epsilon} = {a, d, b, epsilon}

FIRST(S) = FIRST(W) = {a, d, b, epsilon}

FOLLOW(S) = {$}

FOLLOW(W) = {$}

FOLLOW(X) = {c, $}

FOLLOW(Y) = {$}

LL(1) Parsing Table:

Non-Terminal	a	b	c	d	$

S	S -> W	S -> W		S -> W	
W	W -> ZXY	W -> XY		W -> ZXY	
Z	Z -> a			Z -> d	
X	X -> epsilon	X -> Xb	X -> epsilon	X -> epsilon	X -> epsilon
Y	Y -> epsilon		Y -> c		Y -> epsilon

Note: epsilon is included based on FIRST and FOLLOW sets.

LR(1) parsing table:

Grammar:

S -> AA

A -> aA | b

Construct LR(1) items and parsing table using canonical collection of LR(1) items.

(Too large to fit fully here; summarized result below)

States and transitions with lookaheads:

- State I0: S' -> -S, $; S -> -AA, $; A -> -aA, a/b; A -> -b, a/b

Action Table:

State	a	b	$
0	s3	s4	
1			acc
2	s3	s4	

3	s3	s4	
4	r2	r2	r2

... (continues)

Goto Table:

State	S	A
0	1	2
2		5
3		6

...

CLR parsing table:

Grammar:

S -> AA

A -> aA | b

Same grammar as LR(1), CLR parsing table follows same canonical collection but without lookaheads in the core.

Action Table (similar to LR(1), reduced via lookahead-less cores):

[Same entries as above]

Conclusion:

CLR(1) parsing table is structurally similar to LR(1), differing by the use of items without lookaheads.