List of Experiments

S. No.	Topic(s)	CO	PO
1	A. Write a simple calculator program in C/C++/JAVA B. Implementation of basic Flex programs	1	1
2	Implementation of Lexical Analyzer using FLEX.	1	2,5
3	Implementation of calculator using FLEX and BISON.	1	2,5
4	Write a program for Left recursion/Left factoring in C/C++/JAVA	2	2,5
5	Write a program for to Compute FIRST & FOLLOW for Top-Down Parsing and predictive parsing table in C/C++/JAVA	2	2
6	Write a program for Shift Reduce Parsing in C/C++/JAVA	3	2
7	Write a program for Computation of LEADING AND TRAILING in C/C++/JAVA	3	2
8	Write a program for Computation of LR (0) items in C/C++/JAVA	3	2
9	Write an program for Intermediate code generation as Prefix and Suffix in C/C++/JAVA	4	3
10	Write an program for Intermediate code generation as Quadruple, Triple, Indirect triple in C/C++/JAVA	4	3
11	Write a program to generate machine code for a simple statement in C/C++/JAVA	5	3
12	Implement backpatching in C/C++/JAVA	5	3

Procedure

Input: Grammar G
Output: Parsing table

1. Remove left recursion from the grammar G.

a. For each non-terminal A in G, do the following:

i. If there exists a production A -> Aα | β,
where β is not starting with A, split it into:
A -> βA'
A' -> αA' | ε (epsilon), where ε represents the empty string.

ii. If A -> β1 | β2 | ... | βn are the productions of A after step 1,
remove βi if βi starts with A.

2. Compute FIRST sets for each non-terminal and terminal symbol in the grammarG.

a. Initialize FIRST set for each terminal as itself.

b. For each non-terminal A in G, initialize FIRST(A) as an empty set.

c. Repeat until no changes in FIRST sets:

i. For each production A -> α, do the following:

- If α is terminal or ε, add α to FIRST(A).
- If α is non-terminal, add all symbols from FIRST(α) to FIRST(A), except ε.
- If ε is in FIRST(α), continue to the next symbol.

3. Compute FOLLOW sets for each non-terminal in the grammar G.

a. Initialize FOLLOW set for the start symbol S as { $ }, where $ is the end marker.

b. Repeat until no changes in FOLLOW sets:

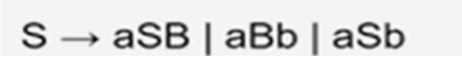
i. For each production A -> αBβ, where B is a non-terminal:

- Add all symbols from FIRST(β) to FOLLOW(B), except ε.
- If β is ε or the symbols in β derive ε, add all symbols from FOLLOW(A) to FOLLOW (B)

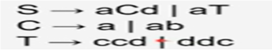
```
4. Construct the parsing table.
   a. Initialize parsing table M with empty entries.
   b. For each production A -> α in G, do the following:
      i. For each terminal a in FIRST(α), add A -> α to M[A, a].
         - If α derives ε, add A -> α to M[A, b] for each terminal b in FOLLOW(A).
      ii. If ε is in FIRST(α), for each terminal b in
          FOLLOW(A), add A -> α to M[A, b].
5. Return the parsing table M.
```

TASK TO BE GIVEN TO THE STUDENTS - KINDLY GIVE THE INPUT CASES DIFFERENTLY FOR EACH STUDENT -REFER BELOW

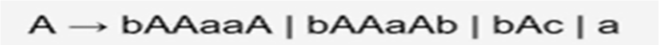
1. **S -> abS | aSb**

2. 

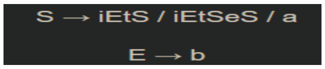
S → aSB | aBb | aSb

3. 

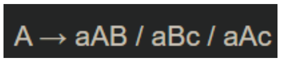
S → aCd | aT
C → a | ab
T → ccd | ddc

4. 

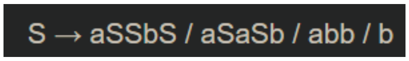
A → bAAaaA | bAAaAb | bAc | a

5. 

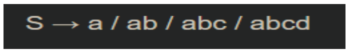
S → iEtS / iEtSeS / a
E → b

6. 

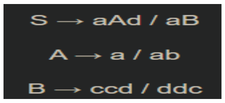
A → aAB / aBc / aAc

7. 

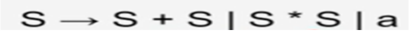
S → aSSbS / aSaSb / abb / b

8. 

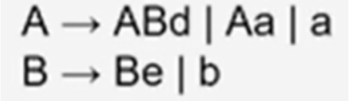
S → a / ab / abc / abcd

9. 

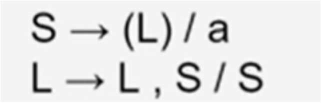
S → aAd / aB
A → a / ab
B → ccd / ddc

10. 

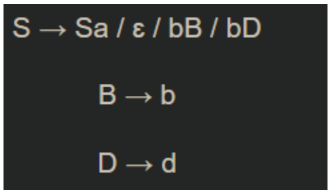
S → S + S | S * S | a

11. 

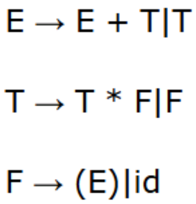
A → ABd | Aa | a
B → Be | b

13. 

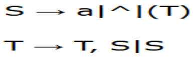
S → (L) / a
L → L , S / S

14. 

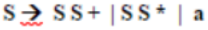
S → Sa / ε / bB / bD
B → b
D → d

15. 

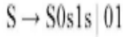
E → E + T|T
T → T * F|F
F → (E)|id

16. 

S → a|^|(T)
T → T, S|S

17. 

S→ SS+ |SS* | a

18. 

S → S0s1s | 01