Lab 1	**Implement file handling operations.**
	Programs Write the following programs: 1. Create, open file and count chars, spaces, tabs, and new lines in a file. 2. Append one file at the end of other. 3. Capitalize the first letter of every word in a file.
Lab 2	**Study Phases of compiler.**
	Examples Explain phases of compiler with output of each phase 1. Position=initial+rate*60 2. A=A+B*C*2 3. F=(9/5)*C+32 4. R=(b*b-4.0*a*c)/(2*a) 5. I=p*n*r/100
Lab 3	**Implement lexical analysis.**
	Programs Write a program to implement lexical analyzer.
Lab 4	**Study LEX tool.**
	Programs Implement following programs using LEX. 1. Write a LEX program to count number of vowels and consonants in each input string. 2. Write a LEX program to take input from text file and count no of characters, no. of lines & no. of words 3. Write a LEX program to identify whether a given line is a comment or not. 4. Implement all the task done by lexical analyzer.
Lab 5	**Study and Implement finite automata.**
	Examples Construct NFA for given regular expression using Thompson's notation and then convert it into DFA. (using Subset construction) 1. (0|1)* 2. (0*|1*)* 3. (a | b)*abb 4. (a+b)*a(a+b) 5. (a+b)*ab*a 6. 0*1*(0|1)# 7. (ab | aa | ba | bb)* 8. a(a+b)*ab **Programs** Write a program to implement DFA for (a+b)*abb

<table>
<tr><td>Lab 6</td><td>Perform String Derivation for a CFG (Context Free Grammar) and Ambiguity</td></tr>
<tr><td></td><td>Examples
Check whether given grammar is ambiguous or not?
1. S→aSbS | bSaS | ε (string: abab)
2. S→A1B
A→0A | ϵ
B→0B | 1B | ϵ (String: 1001)
3. E→E+E | E*E | id | (E) | -E (String: id+id*id)
4. S→ aS | Sa | ϵ (string: aaaa)
Programs
Write a program to check given input string can be derived by following grammar or not.
S→Aa | bAc | dc | bda
A→d</td></tr>
<tr><td>Lab 7</td><td>Perform Left Recursion and Left Factoring</td></tr>
<tr><td></td><td>Examples
Eliminate Left recursion and perform left factoring on given grammar.
Left recursion
1. A→Abd | Aa | a
B→Be | b
2. A→AB | AC | a | b
3. S→A | B
A→ABC | Acd | a | aa
B→Bee | b
4. S→(L) | a
L→L , S | S
Left factoring
1. S→iEtS | iEtSeS | a
2. A→ ad | a | ab | abc | x
3. S→A
A→Ad | Ae | aB | aC
B→bBC | f
C→g
Programs
Write a program to eliminate left recursion from a grammar production.
Write a program to perform left factoring on grammar production.</td></tr>
<tr><td>Lab 8</td><td>Perform LL(1) Parsing method.</td></tr>
<tr><td></td><td>Examples
Perform LL(1) Parsing for the following grammar:
1. S→aBa
B→bB | ϵ
2. check that following grammar is LL(1) or not.
S→aB | ϵ
B→bC | ϵ
C→cS | ϵ
3. Develop an LL(1) parser table for the following grammar and parse the string using the parsing table : (id*id) + (id*id)
E→TA</td></tr>
</table>

	A→+TA| ϵ T→VB B→*VB| ϵ V→id|(E) 4. E→BA A→&BA| ϵ B→true|false C→b 5. S → AaAb | BbBa A→ϵ B→ϵ 6. S→ aAB | bA | ϵ A→ aAb | ϵ B→ bB | ϵ 7. S→iCtSA | a A→ eS | ϵ **Program** Write a program to find FIRST of given grammar.
Lab 9	**Implement Recursive Decent Parsing.**
	Program & Example Write a program to implement recursive decent parser for following grammar: E→E + T | T T→T * F | F F→id
Lab 10	**Implement Operator Precedence Parsing.**
	Examples Check the following grammar is operator grammar or not. Justify your answer. Also prepare operator precedence matrix and graph. 1. E→E + T | T T→T * F | F F→(E) | id 2. E→EAE | id A→+ | * 3. E→EOE E→id O→* | + | - **Program** Write a program to find leading symbol of given production.
Lab 11	**Perform LR Parsing.**
	Examples

	Construct SLR parsing table for given grammar. 1. S→AaBa | BbBa B→Є A→Є 2. E→ E+T | T T→ TF | F F→ F* | a | b Construct CLR parsing table for given grammar. 1. S→CC C→cC | d 2. S→aSA | Є A→bS |c Construct LALR parsing table for given grammar. 1. S→CC C→cC | d 2. S→Aa | bAc | dc | bda A→d
Lab 12	**Study YACC tool.**
	Programs Study about Yet Another Compiler-Compiler (YACC). Create YACC and LEX specification files to generate a calculator which accepts, integer and float type arguments.
Lab 13	**Study SDT (Syntax Directed Translation) for infix to postfix conversion.**
	Examples Write SDT to convert infix to postfix notation and draw annotated parse tree for a given string. 1. 3*5+4n; 2. (3+4)*(5+6)n; 3. 1*2*(3+4)n; **Program** Write a program to convert expression from infix to postfix.
Lab 14	**Study various three address code.**
	Examples Write various representations of three address code for the given expressions: 1. (a = b * -c + b * -c) 2. a*-(b+c) 3. a+a*(b-c)+(b-c)*d 4. ans=a+b*c/2.0 5. x = -a * b + -a * b 6. -(a*b)+(c+d)-(a+b+c+d) **Program** Write a program to read input string from a file and generate three address code.

Lab 15	**Simulation of PDA (Pushdown Automata) and TM (Turing Machine).**
	Examples Design PDA to accept the following languages: 1. L = {x ∈ { a, b}* | $n_a(x) = n_b(x)$}, trace it for string ”abbaba” 2. L= { $a^n b^{n+m} a^m$ | n,m ≥ 0), trace it for string ”abbbaa” 3. L = { xcx^r / x ∈ {a,b}* }, trace it for string “abcba” **Program** Write a program to simulate PDA for language having strings with equal number of a’s and b’s. **Examples** Design TM to accept the following languages: 1. L = { $a^n b^n c^n$ | n ≥ 0 }, trace it for string ”aabbcc” 2. L = { xx^r / x ∈ {a,b}* }, trace it for string “ababa” **Program** Write a program to simulate TM for language L = { $a^n b^n c^n$ | n ≥ 0 }.