Compiler Lab for mid

a) DFA:

1. Design a DFA that accepts all binary strings that end with 01. (Test case: 101,1101,100, 001)
2. Construct a DFA that accepts binary strings containing an even number of 1s. (TC: 1110, 10101, 1100111.
3. Create a DFA that accepts only those strings over {a, b} where no two consecutive b's appear. (TC: abba, ababab, abbba, baaab)

b) NFA to DFA Conversion:

1. Convert the following NFA to DFA:

State	0	1
→A	A, B	A
B	C	B
*C	C	C

2. Given the NFA for the regular expression: (a|b)*abb
3. Convert this ε-NFA to a DFA:

   States: {q0, q1, q2}
   Start: q0
   Final: q2
   Transitions:
   q0 --ε--> q1
   q1 --a--> q1
   q1 --b--> q2

c) Regular Expression to NFA:

1. Construct an NFA for the regular expression: a(b|c)*
2. Convert the regular expression: (ab+b)*a
3. Construct an NFA using **Thompson's** Algorithm for: a*|b(a|b)

d) Implement a Lexical Analyzer:

1. Write a lexical analyzer that identifies the following tokens from input text:

- Keywords: int, float, if, else
- Identifiers
- Numbers
- Operators: + - * / =

  Test on:
  int x = y + 45;

2. Create a lexical analyzer that counts the total number of identifiers in a given C/C++ program.

3. Modify your lexical analyzer to remove comments (// and /* */) and return the cleaned code.

e) Regular Expression to Token Validator:

1. Write a program to validate whether an input string is a valid identifier.

   Rules:

   -Must start with a letter or underscore

   -Can contain letters, digits, and underscores

   -Examples to test: _temp, 9abc, value1, __id.

2. Validate whether a string is a valid integer constant:
   Examples to test: 12, 0034, -45, abc.

3. Validate email format using a regular expression.
   Test input:
   user@gmail.com, test_mail@yahoo, a@b.c.