Q1.

Using the grammar L → id | id , L, where id is an identifier starting with a letter or _ followed by letters, digits, or _, write a C program using recursive descent to check whether a given comma-separated list (e.g., a, a,b,c, x1, y2) is valid.

Q2.

Consider the expression grammar:

- E → T E'
- E' → + T E' | - T E' | ε
- T → F T'
- T' → * F T' | / F T' | ε
- F → (E) | id

Write a C program using recursive descent to check whether a given expression (with identifiers and integers) is syntactically valid.

Q3.

Using the grammar S → (S) S | ε, write a recursive-descent C program that reads a string of parentheses (only (and)) and prints whether it is a valid balanced string according to this grammar.

Q4.

The grammar S → a S b | a b generates strings of the form a^n b^n.

Write a recursive-descent C program that reads a string over {a, b} and checks whether it belongs to this language.

Q5.

Design and implement in C a DFA simulator that accepts all binary strings (over 0 and 1) that end with the substring 01.

The program should read one input string and print “Accepted” if it ends with 01, otherwise “Rejected”.

Q6.

Write a C program to simulate a DFA that accepts all binary strings that contain an even number of 1s.

Use a small transition table (2 states × 2 symbols) and print “Accepted” or “Rejected” for the input string.

Q7.

An NFA has two states: q0 (start) and q1 (accept).

Transitions:

- From q0 on input a → q1

- From q1 on input b or c → q1

This NFA accepts strings of the form a(b|c)*.

Write a C program that simulates this NFA (by tracking the set of current states) and prints whether a given input string is accepted.

Q8.

Write a simple lexical analyzer in C that reads a single line of input and splits it into tokens of three types:

- IDENTIFIER (starting with a letter or _, followed by letters/digits/_)

- NUMBER (sequence of digits)

- SYMBOL (any other single character)

For each token, print its type and lexeme.

Q9.

Write a C program that repeatedly reads words until EOF and classifies each word as:

- a C keyword (from a small fixed list, e.g., int, if, else, for, while, return, char, float, double, void),

- a valid identifier,

- or an invalid token.

Print the classification for each word.

Q10.

Write a C program that reads C source code from standard input and outputs the same code but with all comments removed.

Your program should remove:

- single-line comments starting with // until the end of the line, and

- multi-line comments enclosed by /* and */.