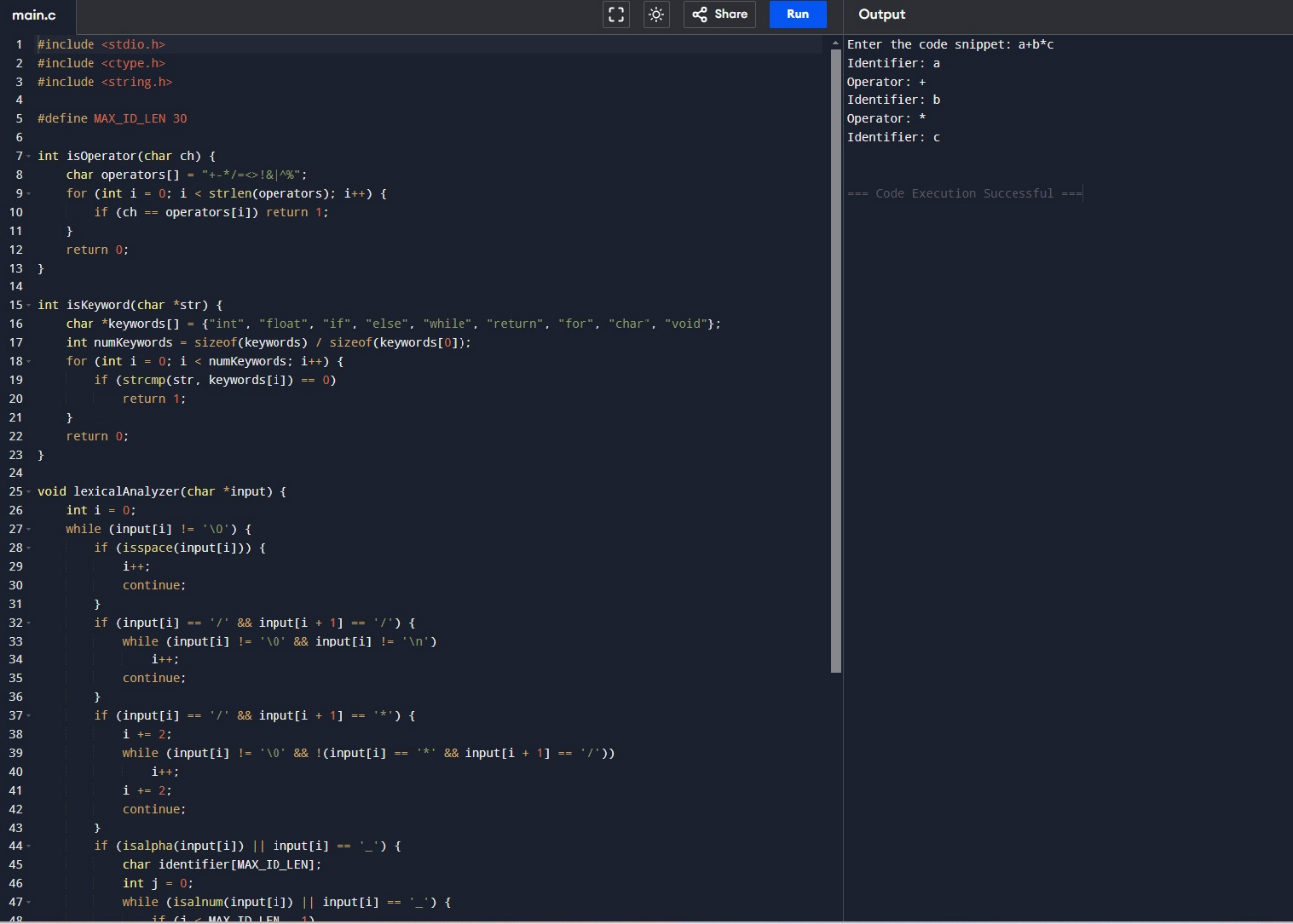
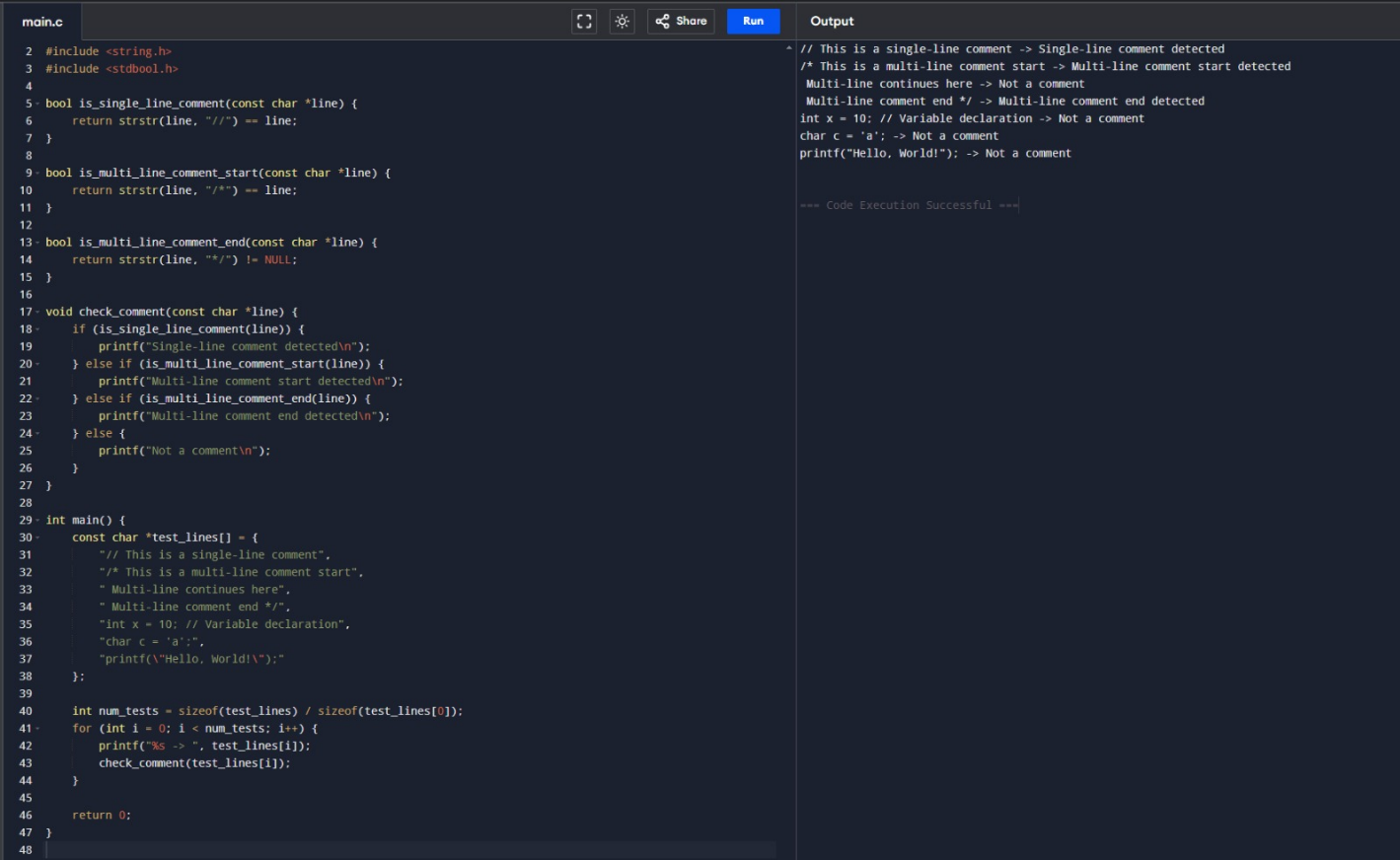
**LIST OF EXPERIMENTS**

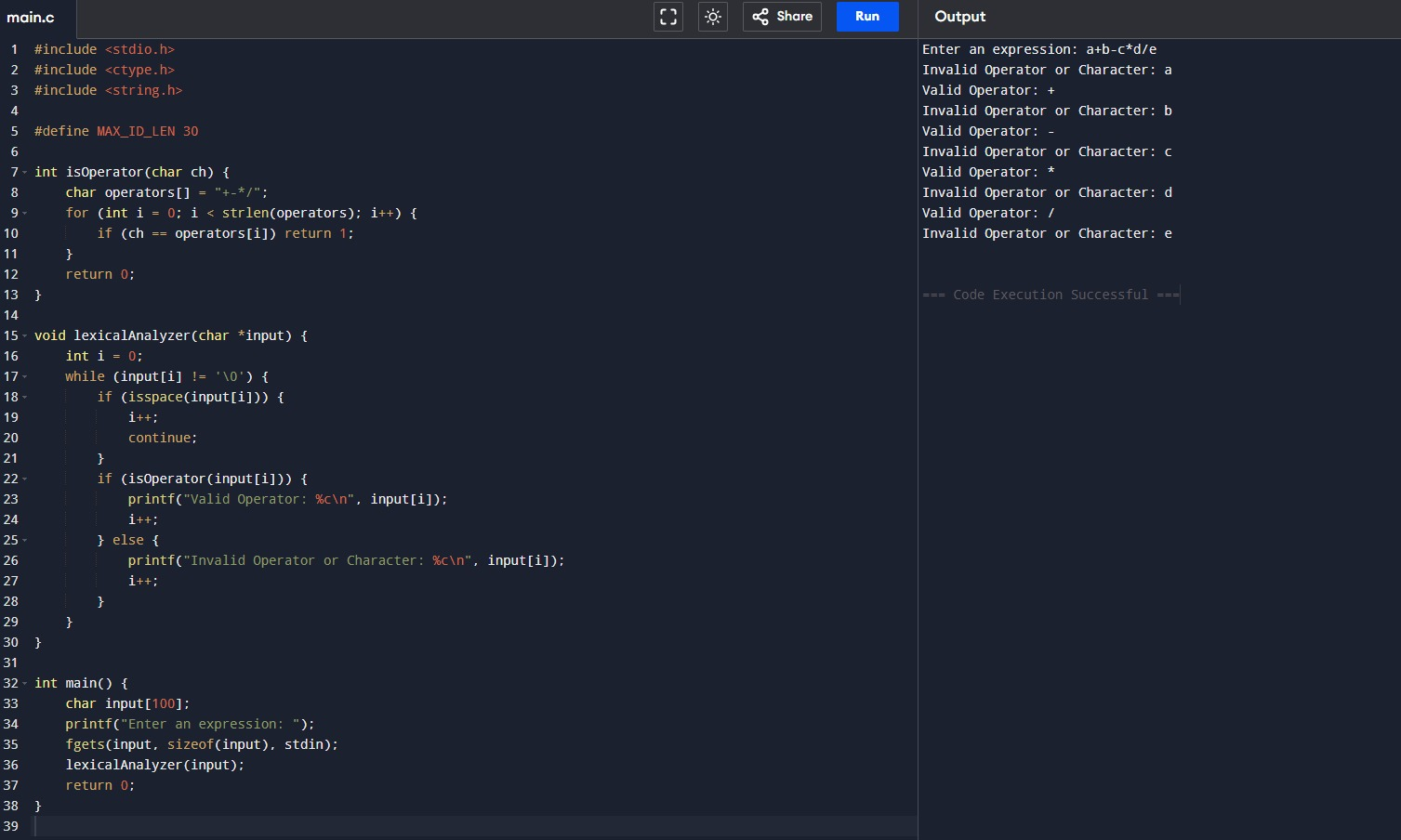
1. The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Develop a lexical Analyzer to identify identifiers, constants, operators using C program.



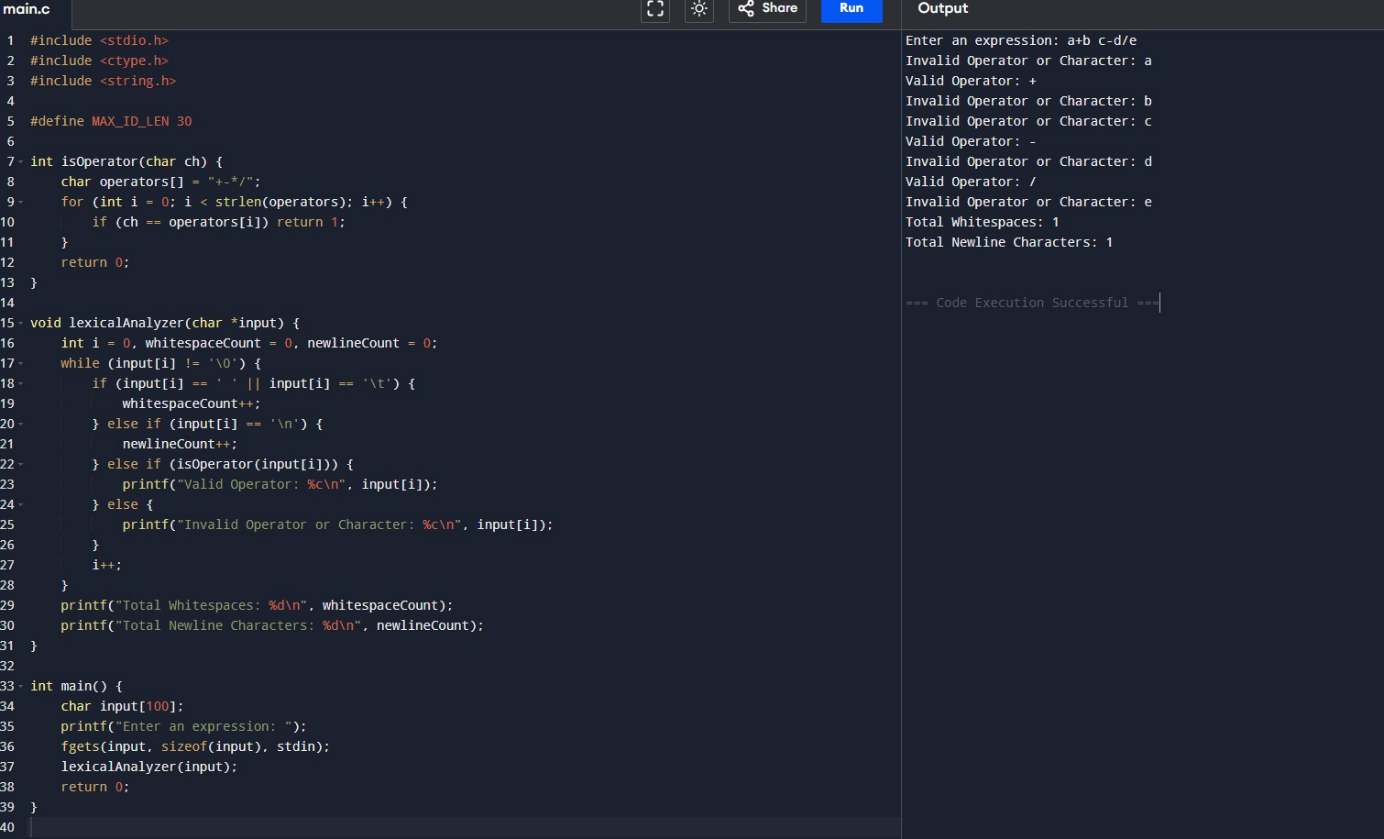
1. Extend the lexical Analyzer to Check comments, dened as follows in C:
   1. A comment begins with // and includes all characters until the end of that line.
   2. A comment begins with /\* and includes all characters through the next occurrence of the character sequence \*/Develop a lexical Analyzer to identify whether a given line is a comment or not.



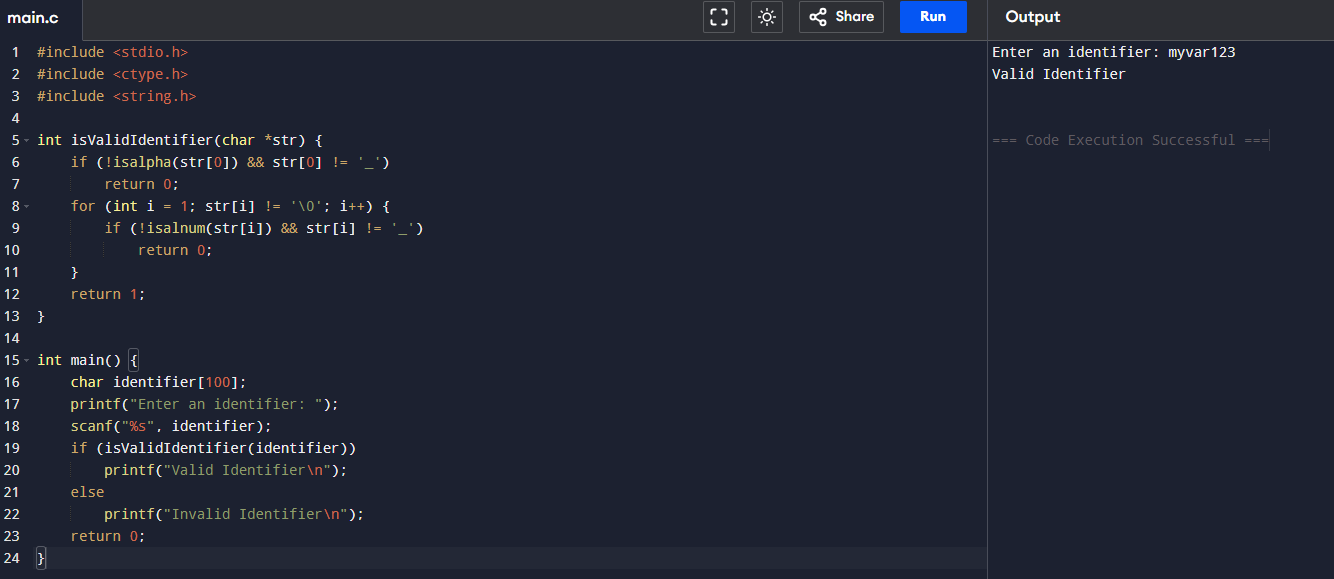
1. Design a lexical Analyzer to validate operators to recognize the operators +,-,\*,/ using regular Arithmetic operators .



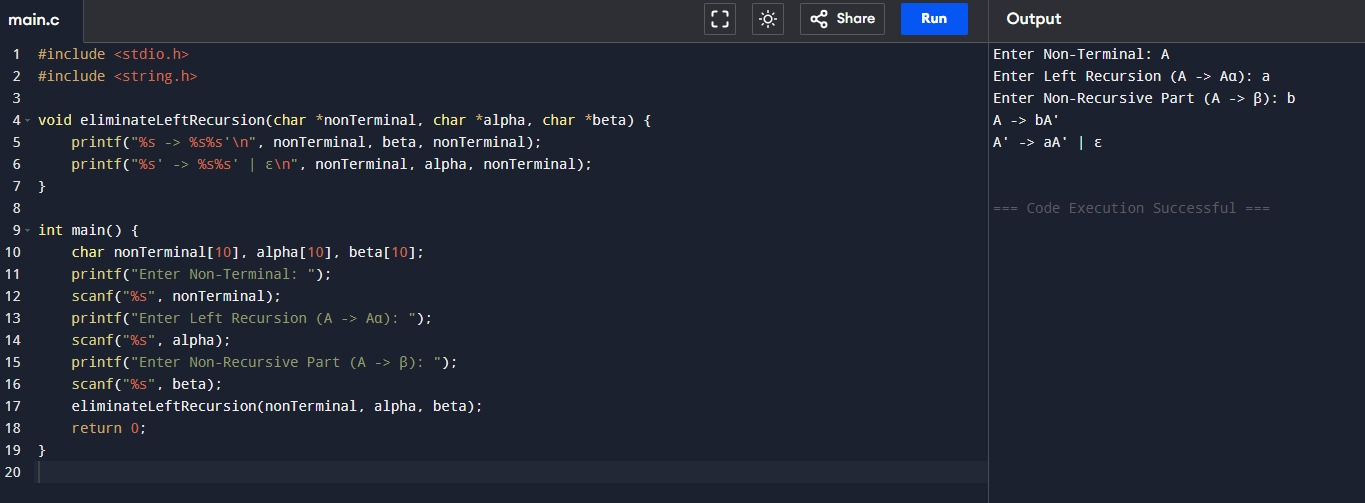
1. Design a lexical Analyzer to find the number of whitespaces and newline characters.



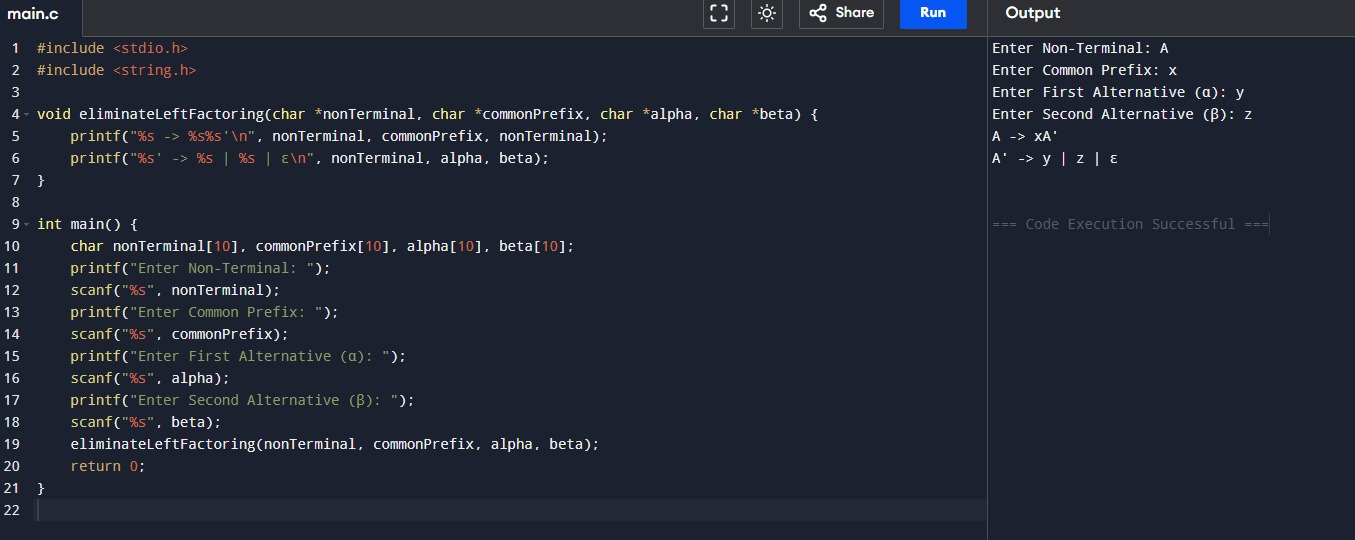
1. Develop a lexical Analyzer to test whether a given identifier is valid or not.

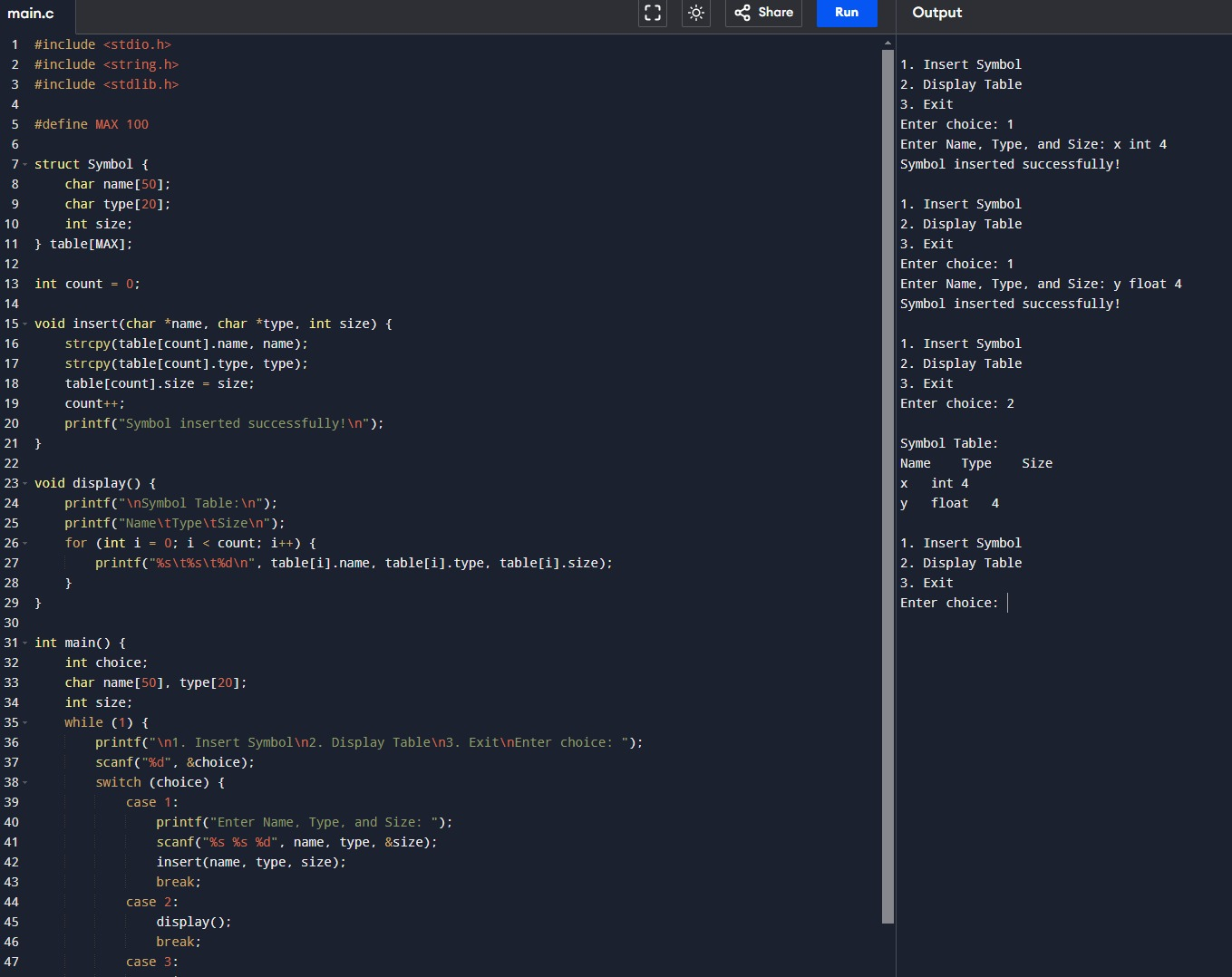


6.Implement a C program to eliminate left recursion

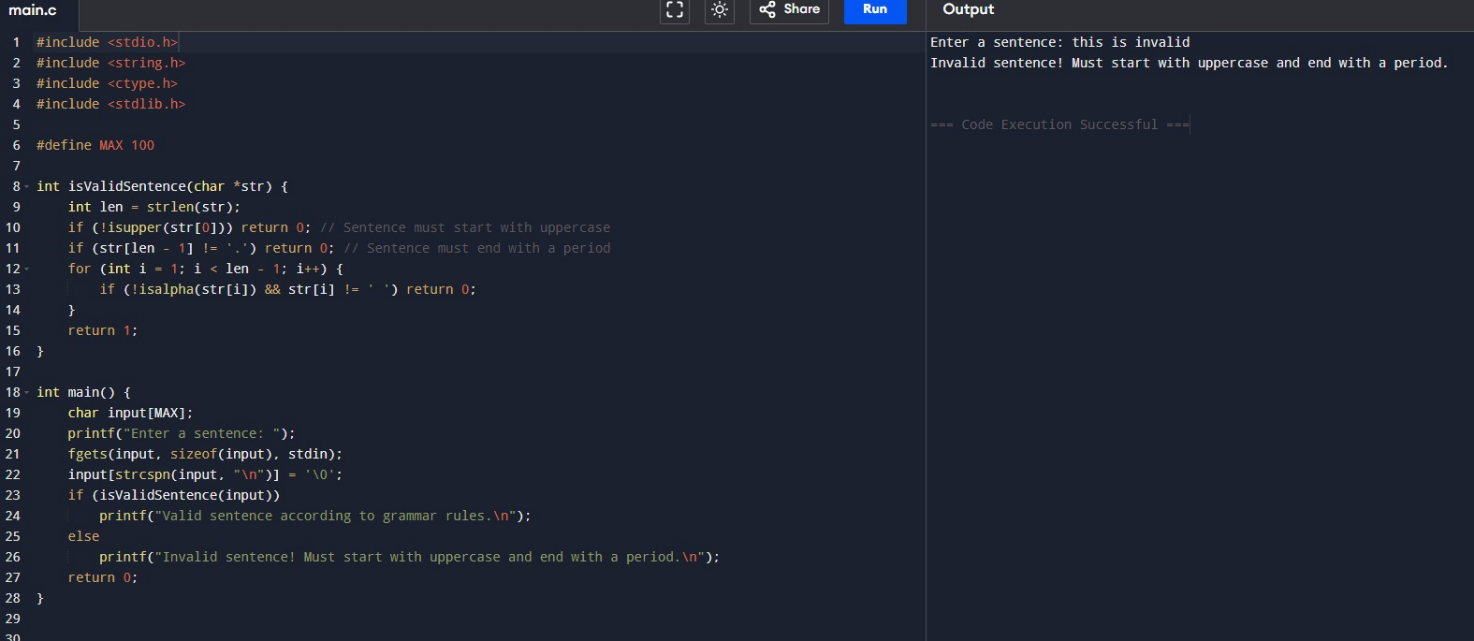


7.Implement a C program to eliminate left factoring.



8.Implement a C program to perform symbol table operations. 

9.All languages have Grammar. When people frame a sentence we usually say whether the sentence is framed as per the rules of the Grammar or Not. Similarly use the same ideology , implement to check whether the given input string is satisfying the grammar or not .



10.Write a C program to construct recursive descent parsing.

