7. Implement a C program to eliminate left factoring.

```c
#include <stdio.h>
#include <string.h>

void eliminateLeftFactoring(char nonTerminal, char alpha[],
    char beta[], char gamma[]) {
    printf("%c -> %s%c'\n", nonTerminal, alpha, nonTerminal);
    printf("%c' -> %s | %s\n", nonTerminal, beta, gamma);
}

int main() {
    char nonTerminal, alpha[10], beta[10], gamma[10];

    printf("Enter non-terminal: ");
    scanf(" %c", &nonTerminal);

    printf("Enter common prefix (alpha): ");
    scanf(" %s", alpha);

    printf("Enter first alternative after prefix (beta): ");
    scanf(" %s", beta);

    printf("Enter second alternative after prefix (gamma): ");
    scanf(" %s", gamma);

    printf("\nGrammar after eliminating left factoring:\n");
    eliminateLeftFactoring(nonTerminal, alpha, beta, gamma);

    return 0;
```

```
Enter non-terminal: H
Enter common prefix (alpha): cf
Enter first alternative after prefix (beta): ad
Enter second alternative after prefix (gamma): be

Grammar after eliminating left factoring:
H -> cfH'
H' -> ad | be

=== Code Execution Successful ===
```

6. Implement a C program to eliminate left recursio

```c
#include <stdio.h>
#include <string.h>

void eliminateLeftRecursion(char nonTerminal, char alpha[], char
    beta[]) {
    printf("%c -> %s%c'\n", nonTerminal, beta, nonTerminal);
    printf("%c' -> %s%c' | ε\n", nonTerminal, alpha, nonTerminal
        );
}

int main() {
    char nonTerminal, alpha[10], beta[10];

    printf("Enter non-terminal: ");
    scanf(" %c", &nonTerminal);

    printf("Enter recursive part (alpha): ");
    scanf(" %s", alpha);

    printf("Enter non-recursive part (beta): ");
    scanf(" %s", beta);

    printf("\nGrammar after eliminating left recursion:\n");
    eliminateLeftRecursion(nonTerminal, alpha, beta);
    return 0;
}
```

```
Enter non-terminal: E
Enter recursive part (alpha): sf
Enter non-recursive part (beta): xc

Grammar after eliminating left recursion:
E -> xcE'
E' -> sfE' | ε

=== Code Execution Successful ===
```

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

Programiz
C Online Compiler

Programiz PRO ›

main.c

Share Run

```c
#include <stdio.h>
#include <ctype.h>

int isValidIdentifier(char *str) {
    if (!isalpha(str[0]) && str[0] != '_') return 0;
    for (int i = 1; str[i] != '\0'; i++) {
        if (!isalnum(str[i]) && str[i] != '_') return 0;
    }
    return 1;
}

int main() {
    char identifier[50];
    printf("Enter an identifier: ");
    scanf("%s", identifier);

    if (isValidIdentifier(identifier))
        printf("Valid Identifier\n");
    else
        printf("Invalid Identifier\n");

    return 0;
}
```

Output

Clear

```
Enter an identifier: my_var3
Valid Identifier


=== Code Execution Successful ===
```


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

Programiz
C Online Compiler

Programiz PRO ›

main.c

Share Run

```c
#include <stdio.h>
#include <string.h>

int main() {
    char input[200];
    int spaces = 0, newlines = 0;

    printf("Enter a multi-line input (end with $ on a new line
        ):\n");

    while (fgets(input, sizeof(input), stdin)) {
        if (input[0] == '$') break;
        for (int i = 0; input[i] != '\0'; i++) {
            if (input[i] == ' ' || input[i] == '\t') spaces++;
            if (input[i] == '\n') newlines++;
        }
    }

    printf("Spaces & tabs: %d\n", spaces);
    printf("Newlines: %d\n", newlines);
    return 0;
}
```

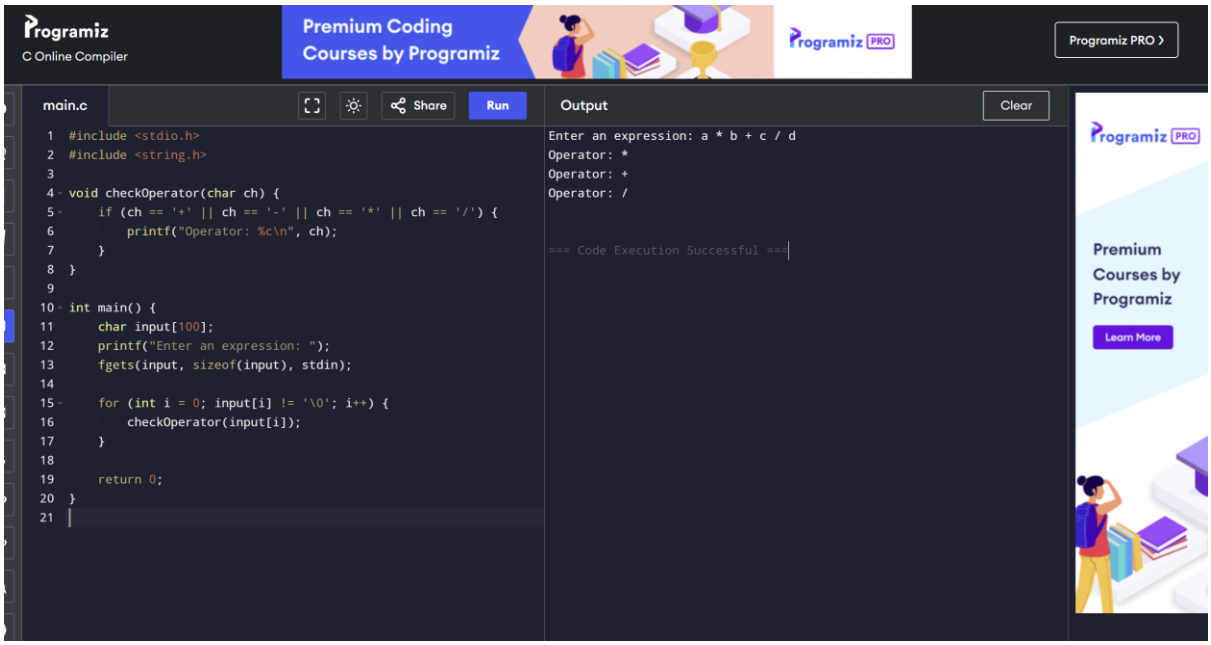
Output

Clear

```
Enter a multi-line input (end with $ on a new line):
int a = 23
float = 4.5
$int a = 11
Spaces & tabs: 5
Newlines: 2


=== Code Execution Successful ===
```

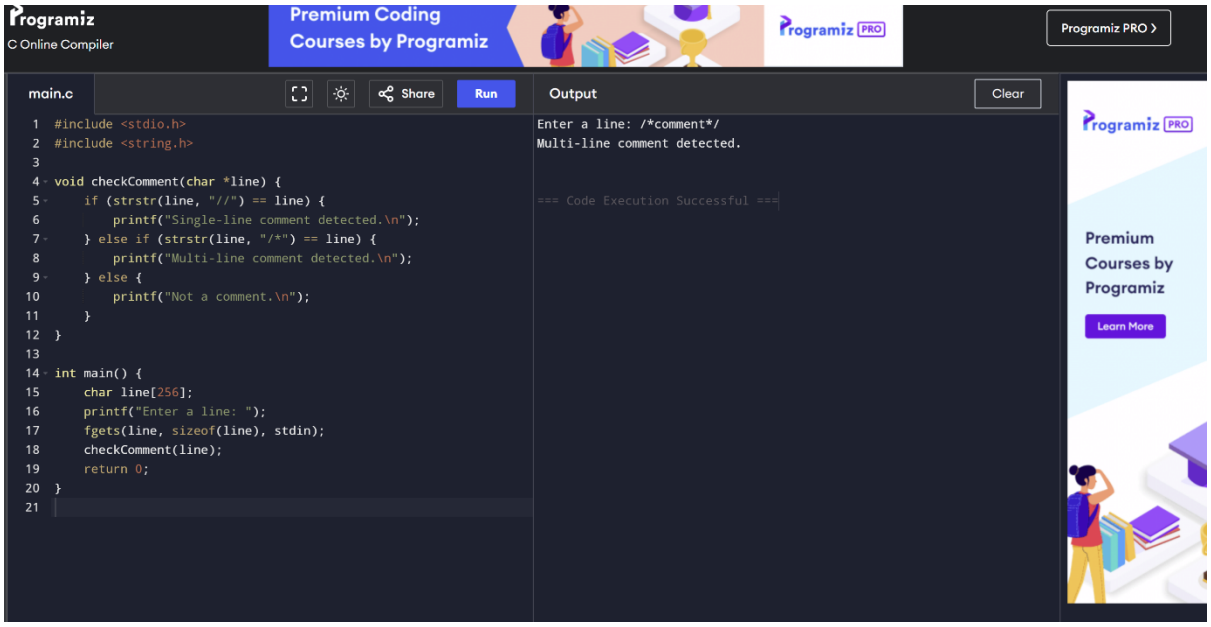
3.Design a lexical Analyzer to validate operators to recognize the operators +,-,*,/ using regular Arithmetic operators



2. Extend the lexical Analyzer to Check comments, dened as follows in C:

a) A comment begins with // and includes all characters until the end of that line.

b) 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. 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.