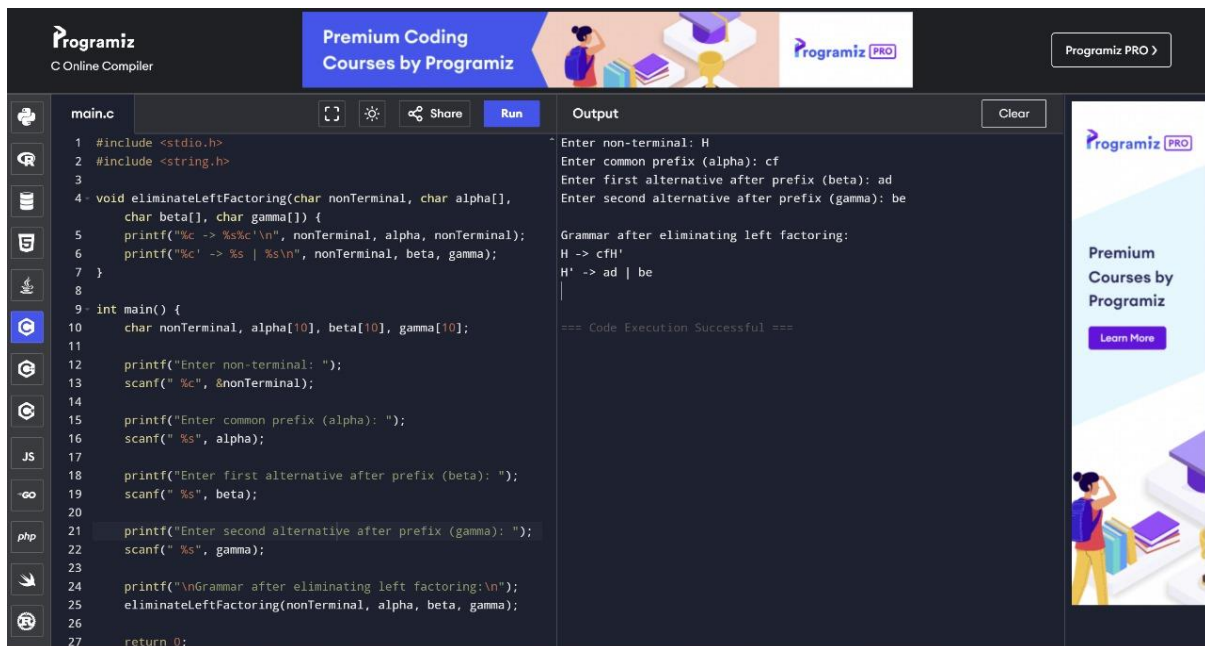


7. Implement a C program to eliminate left factoring.



The screenshot shows the Programiz C Online Compiler interface. The code editor contains a C program for eliminating left factoring. The program prompts the user to enter a non-terminal, a common prefix (alpha), and two alternatives (beta and gamma). It then displays the resulting grammar after eliminating left factoring.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void eliminateLeftFactoring(char nonTerminal, char alpha[],
5                             char beta[], char gamma[]) {
6     printf("%c -> %s%c\n", nonTerminal, alpha, nonTerminal);
7     printf("%c' -> %s | %s\n", nonTerminal, beta, gamma);
8 }
9
10 int main() {
11     char nonTerminal, alpha[10], beta[10], gamma[10];
12
13     printf("Enter non-terminal: ");
14     scanf("%c", &nonTerminal);
15
16     printf("Enter common prefix (alpha): ");
17     scanf("%s", alpha);
18
19     printf("Enter first alternative after prefix (beta): ");
20     scanf("%s", beta);
21
22     printf("Enter second alternative after prefix (gamma): ");
23     scanf("%s", gamma);
24
25     printf("\nGrammar after eliminating left factoring:\n");
26     eliminateLeftFactoring(nonTerminal, alpha, beta, gamma);
27
28     return 0;
29 }
```

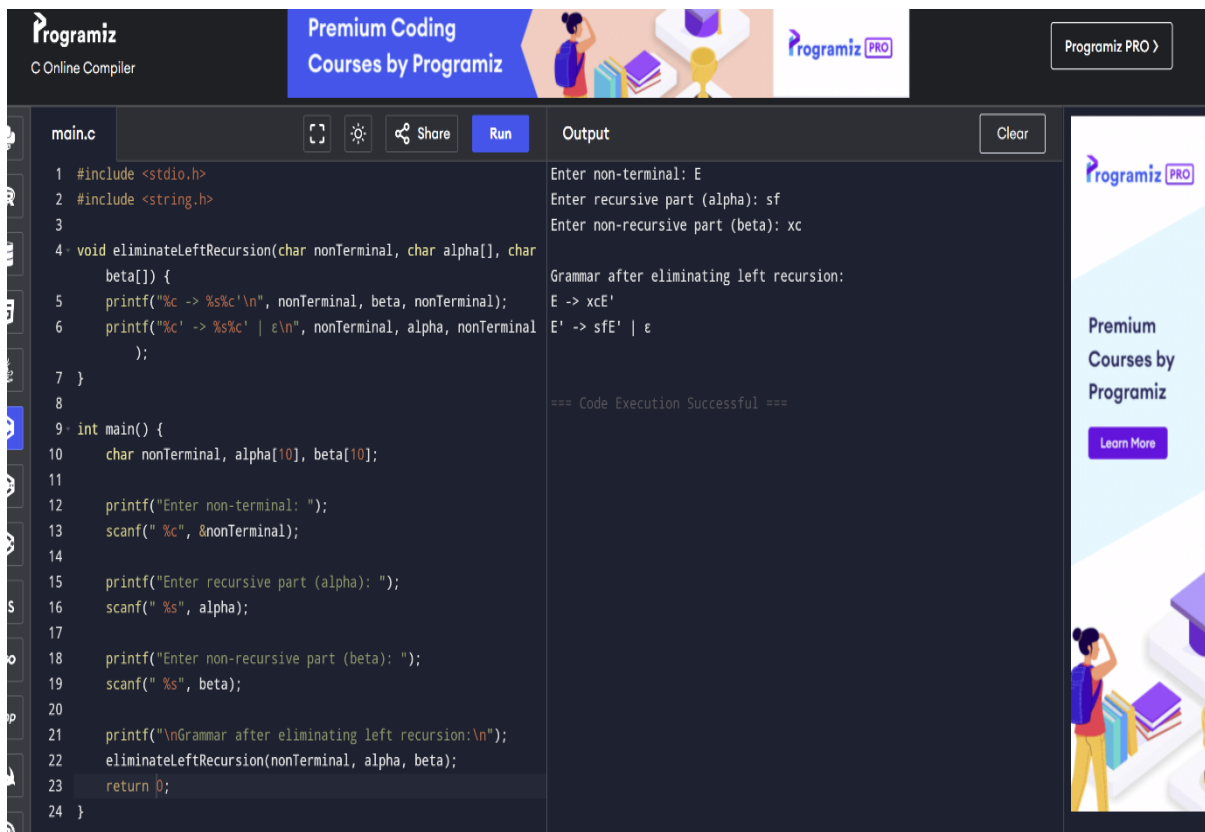
The output shows the following interaction:

```
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 recursion



The screenshot shows the Programiz C Online Compiler interface. The code editor contains a C program for eliminating left recursion. The program prompts the user to enter a non-terminal, a recursive part (alpha), and a non-recursive part (beta). It then displays the resulting grammar after eliminating left recursion.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void eliminateLeftRecursion(char nonTerminal, char alpha[], char
5                             beta[]) {
6     printf("%c -> %s%c\n", nonTerminal, beta, nonTerminal);
7     printf("%c' -> %s%c | ε\n", nonTerminal, alpha, nonTerminal
8 );
9 }
10
11 int main() {
12     char nonTerminal, alpha[10], beta[10];
13
14     printf("Enter non-terminal: ");
15     scanf("%c", &nonTerminal);
16
17     printf("Enter recursive part (alpha): ");
18     scanf("%s", alpha);
19
20     printf("Enter non-recursive part (beta): ");
21     scanf("%s", beta);
22
23     printf("\nGrammar after eliminating left recursion:\n");
24     eliminateLeftRecursion(nonTerminal, alpha, beta);
25
26     return 0;
27 }
```

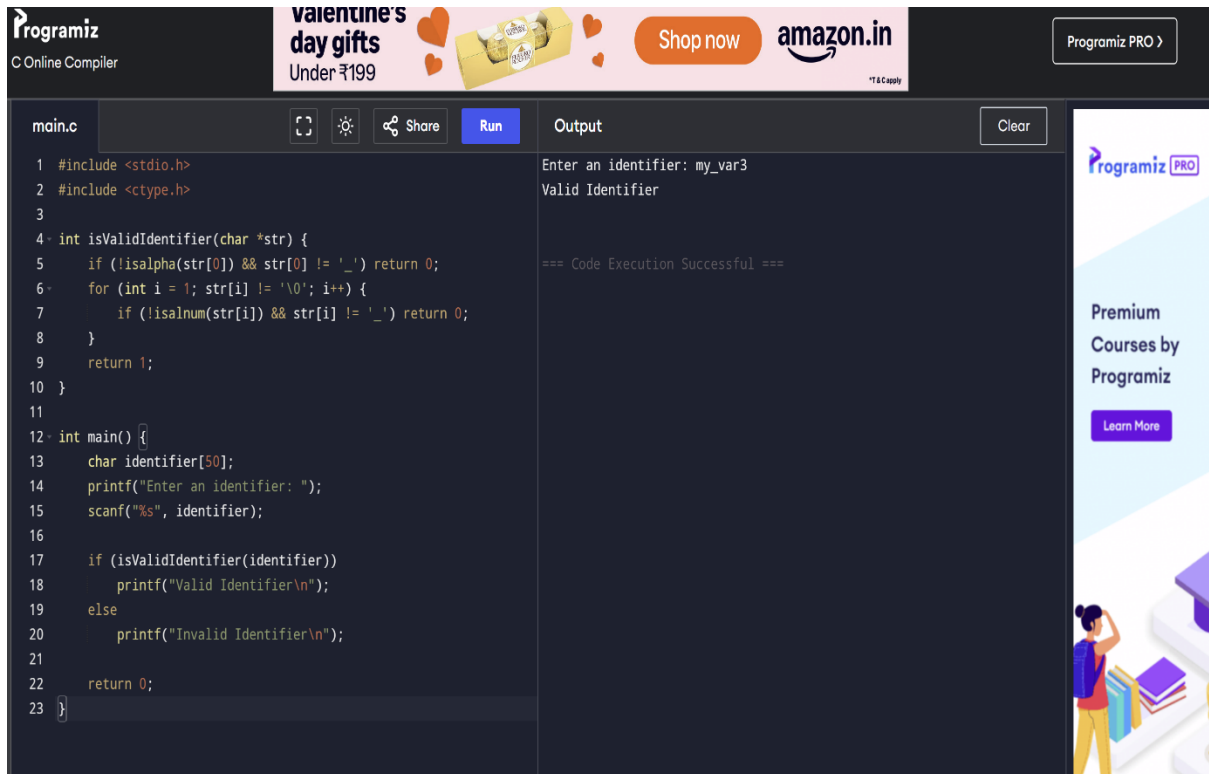
The output shows the following interaction:

```
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 ===
```

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

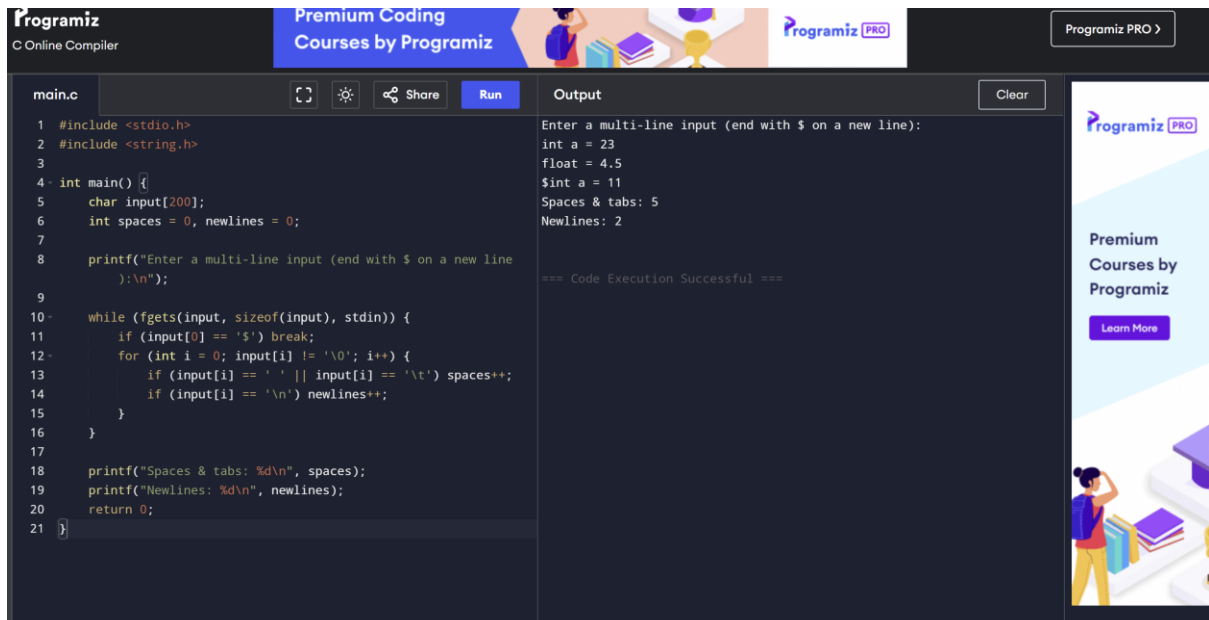


The screenshot shows the Programiz C Online Compiler interface. At the top, there are banners for 'valentine's day gifts' and 'amazon.in'. The main area is divided into a code editor on the left and an output window on the right. The code editor contains a C program that defines a function `isValidIdentifier` to check if a string is a valid identifier. The `main` function prompts the user to enter an identifier and prints the result. The output window shows the user input 'my_var3' and the program's response 'Valid Identifier'.

```
1 #include <stdio.h>
2 #include <ctype.h>
3
4 int isValidIdentifier(char *str) {
5     if (!isalpha(str[0]) && str[0] != '_') return 0;
6     for (int i = 1; str[i] != '\0'; i++) {
7         if (!isalnum(str[i]) && str[i] != '_') return 0;
8     }
9     return 1;
10 }
11
12 int main() {
13     char identifier[50];
14     printf("Enter an identifier: ");
15     scanf("%s", identifier);
16
17     if (isValidIdentifier(identifier))
18         printf("Valid Identifier\n");
19     else
20         printf("Invalid Identifier\n");
21
22     return 0;
23 }
```

Output: Enter an identifier: my_var3
Valid Identifier
=== Code Execution Successful ===

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

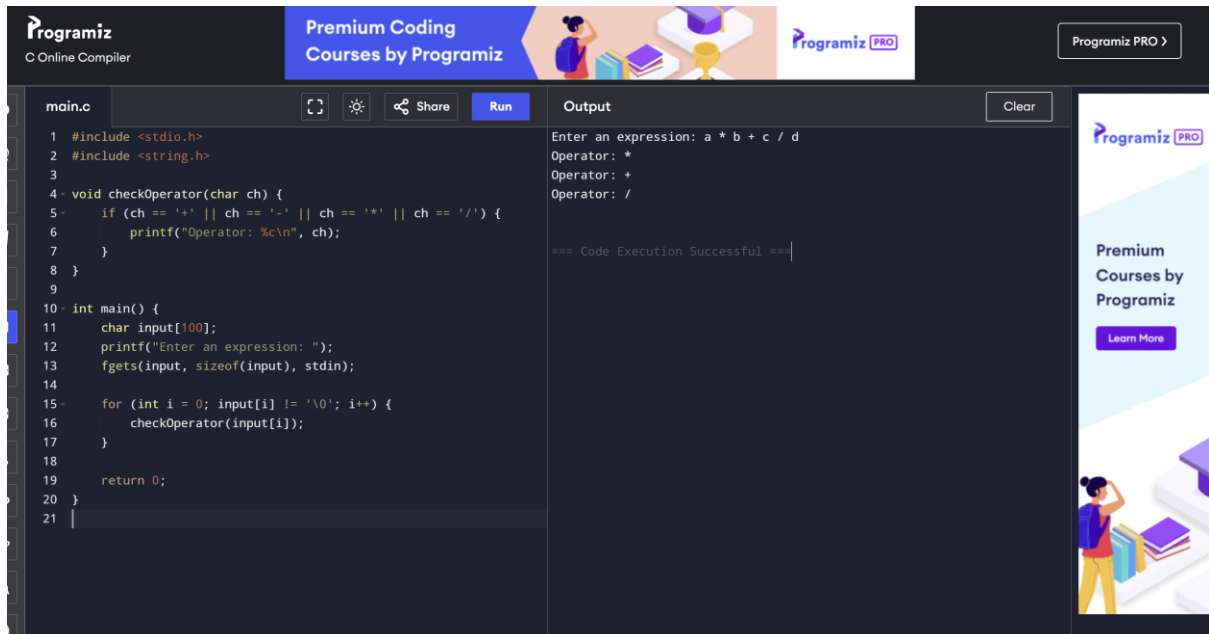


The screenshot shows the Programiz C Online Compiler interface. At the top, there are banners for 'Premium Coding Courses by Programiz' and 'Programiz PRO'. The main area is divided into a code editor on the left and an output window on the right. The code editor contains a C program that prompts the user to enter a multi-line input. The program then counts the number of spaces and tabs, and the number of newlines in the input. The output window shows the user input and the program's response.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char input[200];
6     int spaces = 0, newlines = 0;
7
8     printf("Enter a multi-line input (end with $ on a new line):\n");
9
10    while (fgets(input, sizeof(input), stdin)) {
11        if (input[0] == '$') break;
12        for (int i = 0; input[i] != '\0'; i++) {
13            if (input[i] == ' ' || input[i] == '\t') spaces++;
14            if (input[i] == '\n') newlines++;
15        }
16    }
17
18    printf("Spaces & tabs: %d\n", spaces);
19    printf("Newlines: %d\n", newlines);
20    return 0;
21 }
```

Output: 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



The screenshot displays the Programiz C Online Compiler interface. The top header includes the Programiz logo, a navigation bar with 'Premium Coding Courses by Programiz', and a 'Programiz PRO' badge. The main workspace is divided into two panels: a code editor on the left and an output console on the right. The code editor shows a C program named 'main.c' that defines a function 'checkOperator' to validate arithmetic operators (+, -, *, /) and a 'main' function that prompts the user to enter an expression and iterates through each character to check if it is a valid operator. The output console shows the program's execution for the input 'a * b + c / d', displaying the detected operators: '*', '+', and '/'. A 'Clear' button is located in the top right of the output panel. On the right side of the interface, there is a vertical sidebar with the Programiz logo, the text 'Premium Courses by Programiz', a 'Learn More' button, and an illustration of a person with a backpack.

```
main.c
1 #include <stdio.h>
2 #include <string.h>
3
4 void checkOperator(char ch) {
5     if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
6         printf("Operator: %c\n", ch);
7     }
8 }
9
10 int main() {
11     char input[100];
12     printf("Enter an expression: ");
13     fgets(input, sizeof(input), stdin);
14
15     for (int i = 0; input[i] != '\0'; i++) {
16         checkOperator(input[i]);
17     }
18
19     return 0;
20 }
21
```

Output

Enter an expression: a * b + c / d
Operator: *
Operator: +
Operator: /

=== Code Execution Successful ===

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

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

Programiz
C Online Compiler

Premium Coding
Courses by Programiz

Programiz PRO

Programiz PRO >

main.c

Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void checkComment(char *line) {
5     if (strstr(line, "/*") == line) {
6         printf("Single-line comment detected.\n");
7     } else if (strstr(line, "/*") == line) {
8         printf("Multi-line comment detected.\n");
9     } else {
10        printf("Not a comment.\n");
11    }
12 }
13
14 int main() {
15     char line[256];
16     printf("Enter a line: ");
17     fgets(line, sizeof(line), stdin);
18     checkComment(line);
19     return 0;
20 }
21
```


Enter a line: /*comment*/
Multi-line comment detected.

=== Code Execution Successful ===

Programiz PRO


Premium
Courses by
Programiz

Learn More



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.

Programiz
C Online Compiler



Grains Are Back in the Kitchen—And He's Not Talking
Wheat and Rice
Sponsored by: Mansion Global

Learn More

Programiz PRO >

main.c

Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <ctype.h>
3 #include <string.h>
4
5 #define MAX_LEN 100
6
7 void checkToken(char *token) {
8     if (isdigit(token[0])) {
9         printf("CONSTANT: %s\n", token);
10    } else if (isalpha(token[0]) || token[0] == '_') {
11        printf("IDENTIFIER: %s\n", token);
12    } else if (strchr("+-*/=<>!", token[0])) {
13        printf("OPERATOR: %s\n", token);
14    }
15 }
16
17 int main() {
18     char input[MAX_LEN], buffer[MAX_LEN];
19     int i = 0;
20
21     printf("Enter a line of code: ");
22     fgets(input, MAX_LEN, stdin);
23
24     for (int j = 0; input[j] != '\0'; j++) {
25         if (isspace(input[j])) {
26             if (i != 0) {
27                 buffer[i] = '\0';
28                 checkToken(buffer);
29                 i = 0;
30             }
31         } else {
32             buffer[i] = input[j];
33             i++;
34         }
35     }
36     buffer[i] = '\0';
37     checkToken(buffer);
38 }
```

Enter a line of code: y=10+h
IDENTIFIER: y
OPERATOR: =
CONSTANT: 10
OPERATOR: +
IDENTIFIER: h

=== Code Execution Successful ===

Programiz PRO

Premium
Courses by
Programiz

Learn More

