

LAB

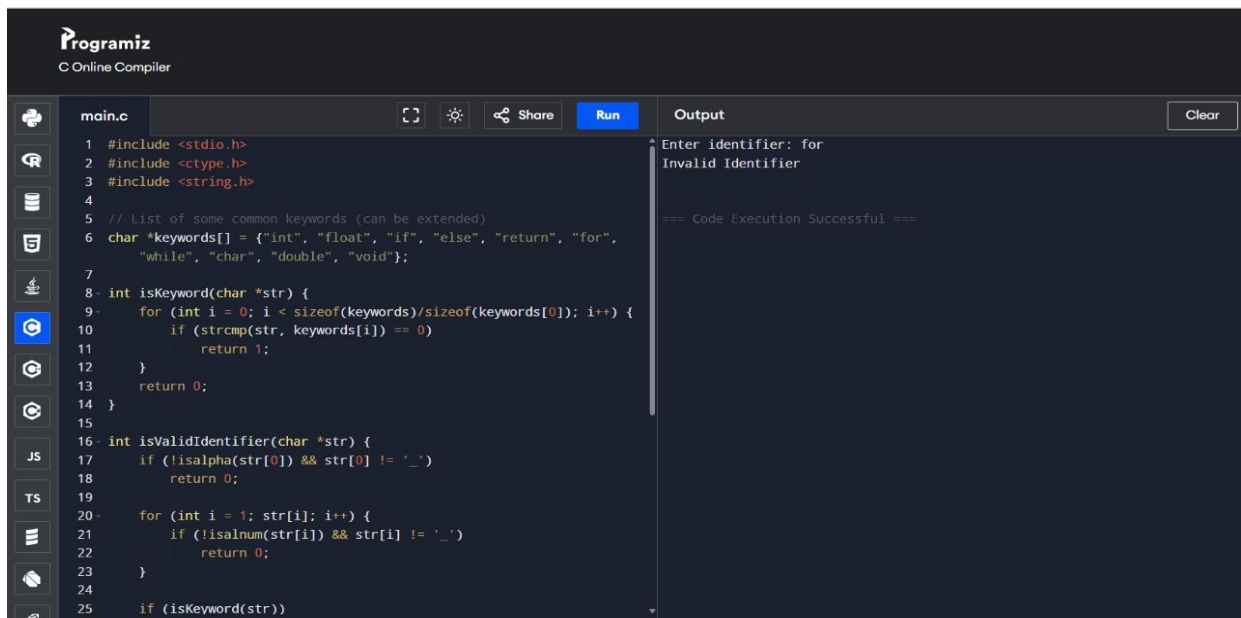
NAME : ASWANTH N N

REG NO : 192424123

COURSE : CSA1423-Compiler Design

EXP 3

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



The screenshot displays the Programiz C Online Compiler interface. The code editor on the left contains a C program named `main.c` that implements a lexical analyzer to check if a given string is a valid identifier. The program includes standard headers, a list of common keywords, and two functions: `isKeyword` and `isValidIdentifier`. The `isValidIdentifier` function checks if the first character is an alphabetic character or an underscore, and if subsequent characters are alphanumeric or underscores. The `main` function calls `isValidIdentifier` with the input string `"for"`. The output panel on the right shows the execution results.

```
1 #include <stdio.h>
2 #include <ctype.h>
3 #include <string.h>
4
5 // List of some common keywords (can be extended)
6 char *keywords[] = {"int", "float", "if", "else", "return", "for",
7                     "while", "char", "double", "void"};
8
9 int isKeyword(char *str) {
10     for (int i = 0; i < sizeof(keywords)/sizeof(keywords[0]); i++) {
11         if (strcmp(str, keywords[i]) == 0)
12             return 1;
13     }
14     return 0;
15 }
16
17 int isValidIdentifier(char *str) {
18     if (!isalpha(str[0]) && str[0] != '_')
19         return 0;
20
21     for (int i = 1; str[i]; i++) {
22         if (!isalnum(str[i]) && str[i] != '_')
23             return 0;
24     }
25
26     if (isKeyword(str))
```

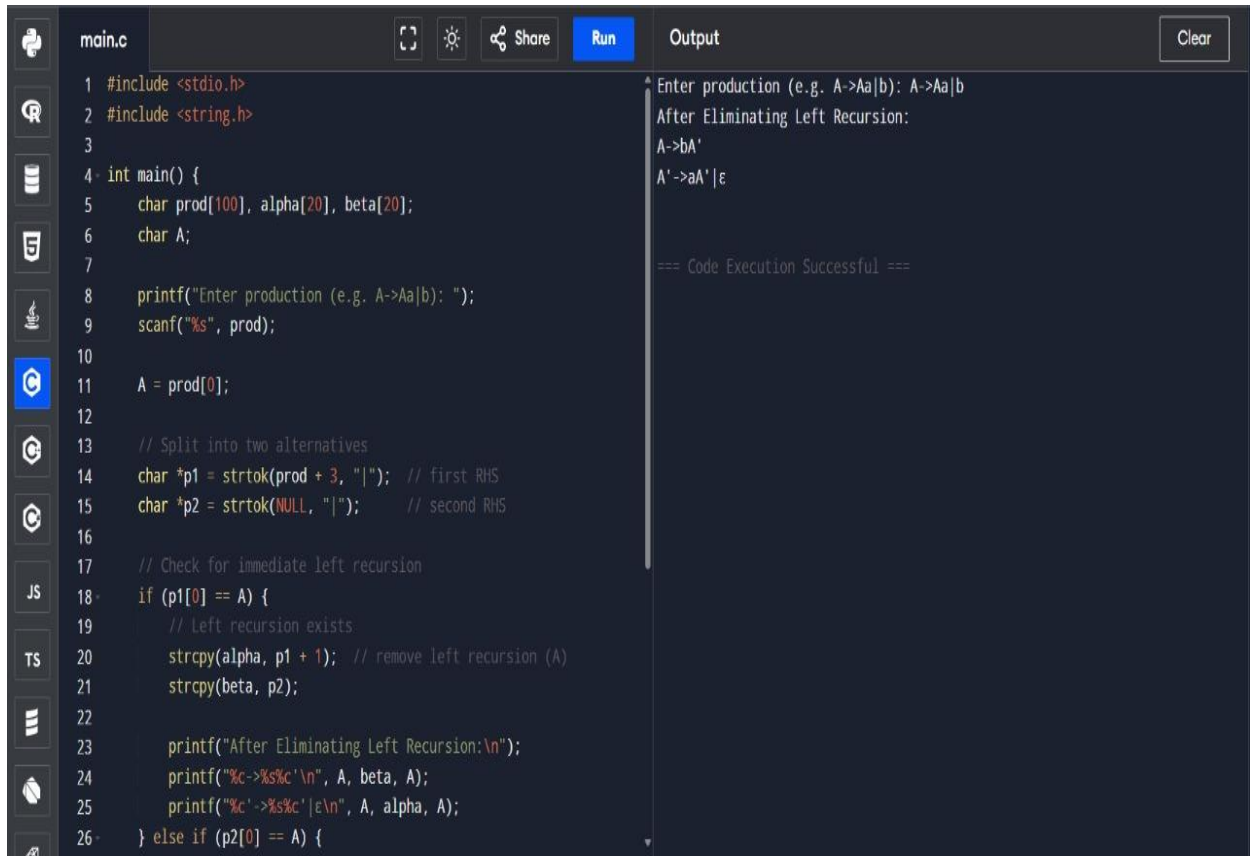
Output:

```
Enter identifier: for
Invalid Identifier

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

EXP 4

Implement a C program to eliminate left recursion.



```
main.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char prod[100], alpha[20], beta[20];
6     char A;
7
8     printf("Enter production (e.g. A->Aa|b): ");
9     scanf("%s", prod);
10
11     A = prod[0];
12
13     // Split into two alternatives
14     char *p1 = strtok(prod + 3, "|"); // first RHS
15     char *p2 = strtok(NULL, "|");    // second RHS
16
17     // Check for immediate left recursion
18     if (p1[0] == A) {
19         // Left recursion exists
20         strcpy(alpha, p1 + 1); // remove left recursion (A)
21         strcpy(beta, p2);
22
23         printf("After Eliminating Left Recursion:\n");
24         printf("%c->%s%c'\n", A, beta, A);
25         printf("%c'->%s%c'|\n", A, alpha, A);
26     } else if (p2[0] == A) {
```

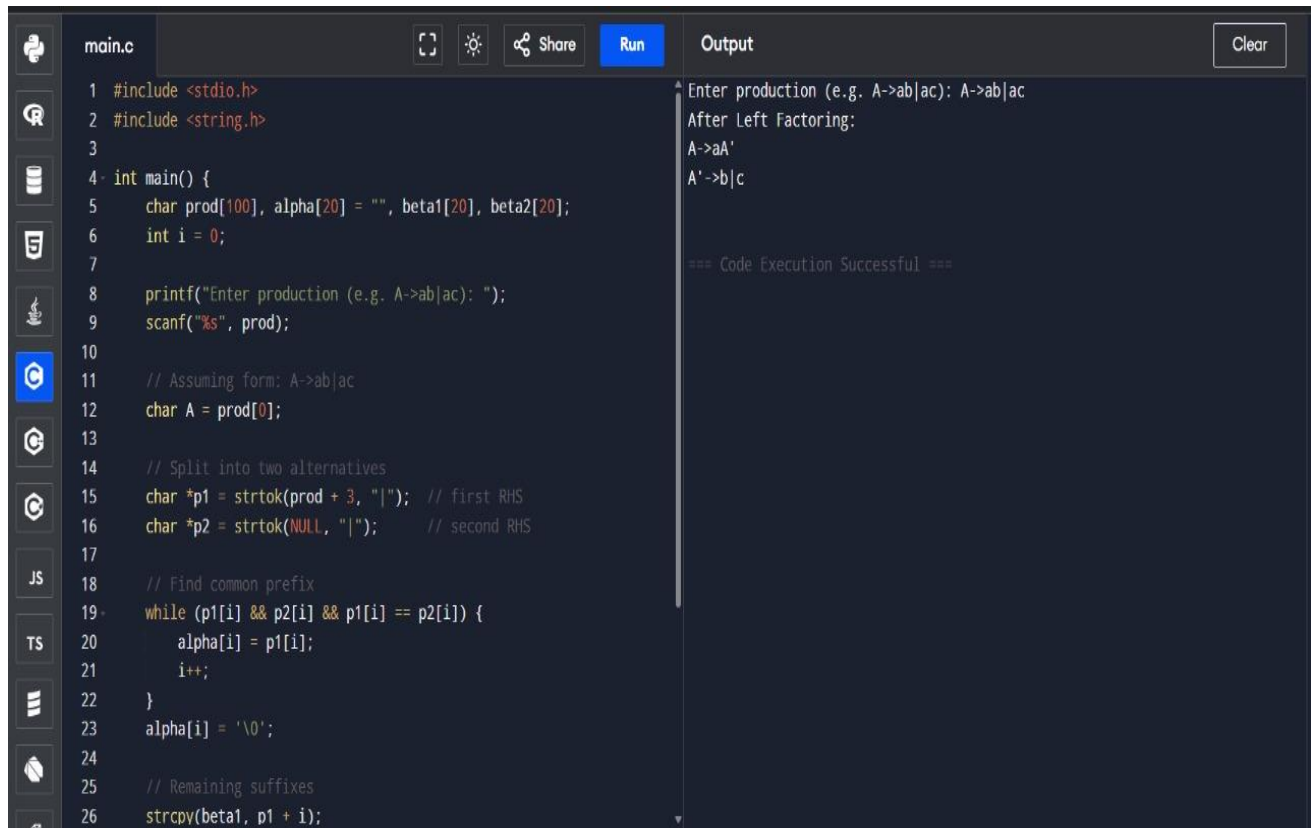
Output

```
Enter production (e.g. A->Aa|b): A->Aa|b
After Eliminating Left Recursion:
A->bA'
A'->aA'|\n

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

EXP 5

Implement a C program to eliminate left factoring.



The screenshot shows a code editor with a C program for left factoring. The code is as follows:

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char prod[100], alpha[20] = "", beta1[20], beta2[20];
6     int i = 0;
7
8     printf("Enter production (e.g. A->ab|ac): ");
9     scanf("%s", prod);
10
11     // Assuming form: A->ab|ac
12     char A = prod[0];
13
14     // Split into two alternatives
15     char *p1 = strtok(prod + 3, "|"); // first RHS
16     char *p2 = strtok(NULL, "|");    // second RHS
17
18     // Find common prefix
19     while (p1[i] && p2[i] && p1[i] == p2[i]) {
20         alpha[i] = p1[i];
21         i++;
22     }
23     alpha[i] = '\0';
24
25     // Remaining suffixes
26     strcpy(beta1, p1 + i);
```

The output window shows the following text:

```
Enter production (e.g. A->ab|ac): A->ab|ac
After Left Factoring:
A->aA'
A'->b|c

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