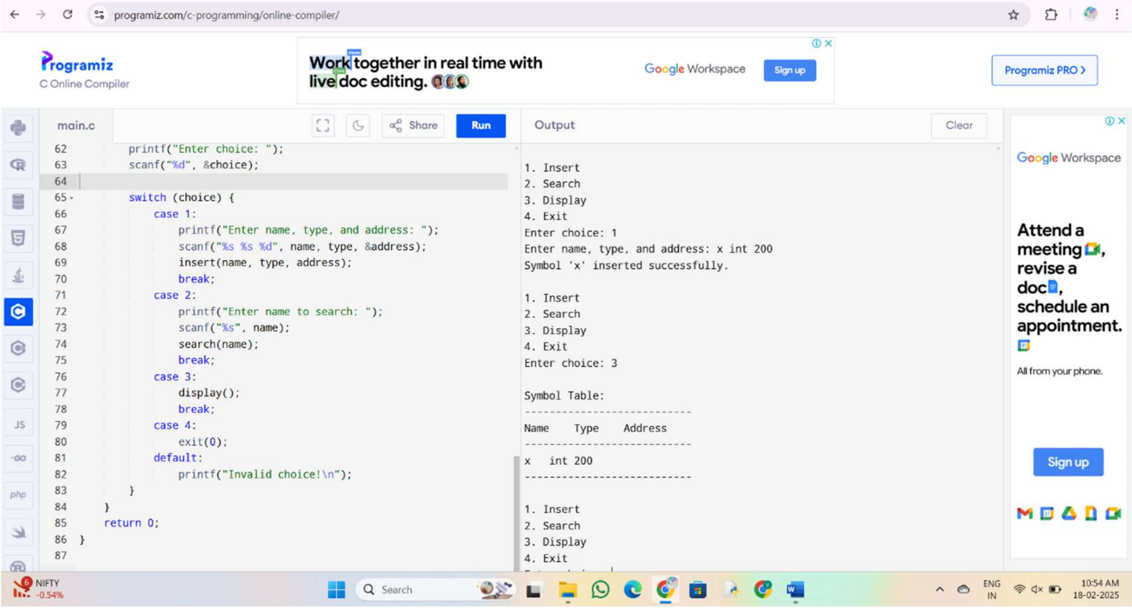
8. Implement a C program to perform symbol table operations.

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

#define MAX 100

typedef struct {
    char name[50];
    char type[20];
    int address;
} Symbol;

Symbol table[MAX];
int count = 0;

void insert(char name[], char type[], int address) {
    for (int i = 0; i < count; i++) {
        if (strcmp(table[i].name, name) == 0) {
            printf("Error: Symbol '%s' already exists!\n", name);
            return;
        }
    }
    strcpy(table[count].name, name);
    strcpy(table[count].type, type);
    table[count].address = address;
    count++;
    printf("Symbol '%s' inserted successfully.\n", name);
}

void search(char name[]) {
    for (int i = 0; i < count; i++) {
        if (strcmp(table[i].name, name) == 0) {
            printf("Symbol Found: Name=%s, Type=%s, Address=%d\n", table[i].name,
table[i].type, table[i].address);
            return;
        }
    }
    printf("Symbol '%s' not found!\n", name);
}

void display() {
    if (count == 0) {
        printf("Symbol table is empty!\n");
        return;
    }
    printf("\nSymbol Table:\n");
    printf("---------------------------\n");
    printf("Name\tType\tAddress\n");
    printf("---------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("%s\t%s\t%d\n", table[i].name, table[i].type, table[i].address);
    }
    printf("---------------------------\n");
```

```
}

int main() {
    int choice;
    char name[50], type[20];
    int address;

    while (1) {
        printf("\n1. Insert\n2. Search\n3. Display\n4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter name, type, and address: ");
                scanf("%s %s %d", name, type, &address);
                insert(name, type, address);
                break;
            case 2:
                printf("Enter name to search: ");
                scanf("%s", name);
                search(name);
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:

                printf("Invalid choice!\n");

        }

    }

    return 0;

}
```

Op:



```
    printf("Enter choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter name, type, and address: ");
            scanf("%s %s %d", name, type, &address);
            insert(name, type, address);
            break;
        case 2:
            printf("Enter name to search: ");
            scanf("%s", name);
            search(name);
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice!\n");
    }
}
return 0;
}
```

```
1. Insert
2. Search
3. Display
4. Exit
Enter choice: 1
Enter name, type, and address: x int 200
Symbol 'x' inserted successfully.

1. Insert
2. Search
3. Display
4. Exit
Enter choice: 3

Symbol Table:
--------------------------
Name    Type    Address
--------------------------
x   int 200
--------------------------

1. Insert
2. Search
3. Display
4. Exit
```

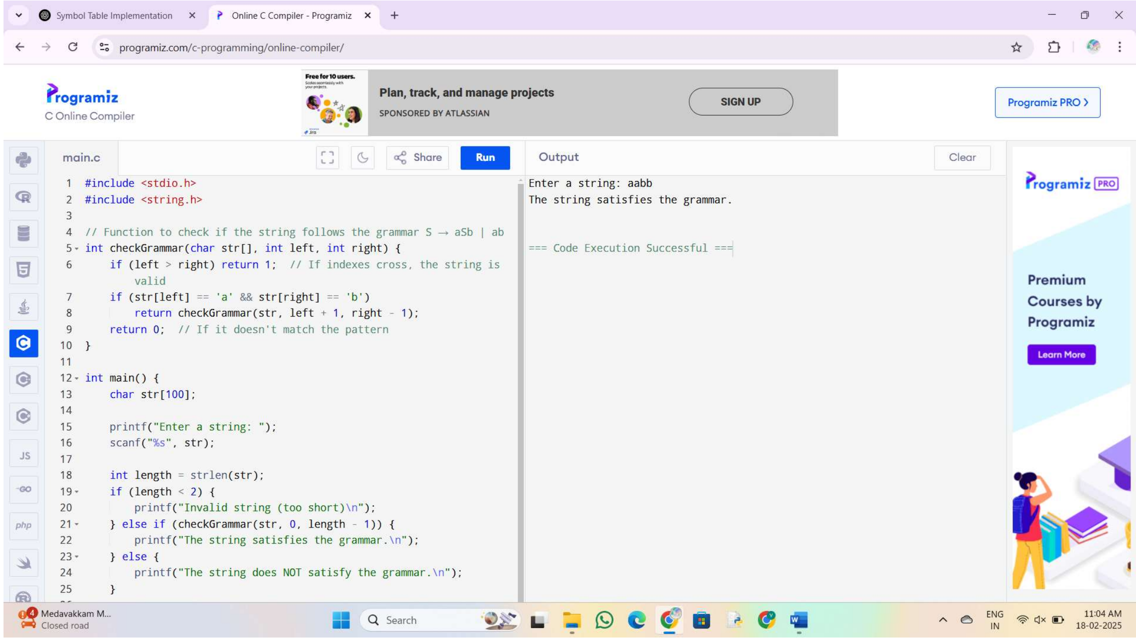
9. All languages have Grammar. When people frame a sentence we usually say whether the sentence is framed as per the rules of the Grammar or Not. Similarly use the same ideology , implement to check whether the given input string is satisfying the grammar or not .

Code:

```
#include <stdio.h>

#include <string.h>


// Function to check if the string follows the grammar S → aSb | ab

int checkGrammar(char str[], int left, int right) {

    if (left > right) return 1;  // If indexes cross, the string is valid

    if (str[left] == 'a' && str[right] == 'b')
```

```
        return checkGrammar(str, left + 1, right - 1);
    return 0;  // If it doesn't match the pattern
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%s", str);

    int length = strlen(str);
    if (length < 2) {
        printf("Invalid string (too short)\n");
    } else if (checkGrammar(str, 0, length - 1)) {
        printf("The string satisfies the grammar.\n");
    } else {
        printf("The string does NOT satisfy the grammar.\n");
    }

    return 0;
}
```

Op:


Symbol Table Implementation
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Plan, track, and manage projects
SPONSORED BY ATLASSIAN
SIGN UP
Programiz PRO >
main.c
Share
Run
Output
Clear
Premium Courses by Programiz
Learn More
Medavakkam M...
Closed road
Search
ENG
IN
11:04 AM
18-02-2025

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

// Function to check if the string follows the grammar S → aSb | ab
int checkGrammar(char str[], int left, int right) {
    if (left > right) return 1;  // If indexes cross, the string is
        valid
    if (str[left] == 'a' && str[right] == 'b')
        return checkGrammar(str, left + 1, right - 1);
    return 0;  // If it doesn't match the pattern
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%s", str);

    int length = strlen(str);
    if (length < 2) {
        printf("Invalid string (too short)\n");
    } else if (checkGrammar(str, 0, length - 1)) {
        printf("The string satisfies the grammar.\n");
    } else {
        printf("The string does NOT satisfy the grammar.\n");
    }
```

```
Enter a string: aabb
The string satisfies the grammar.


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

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

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char input[100];

int pos = 0;

// Function Prototypes

void E();

void Eprime();

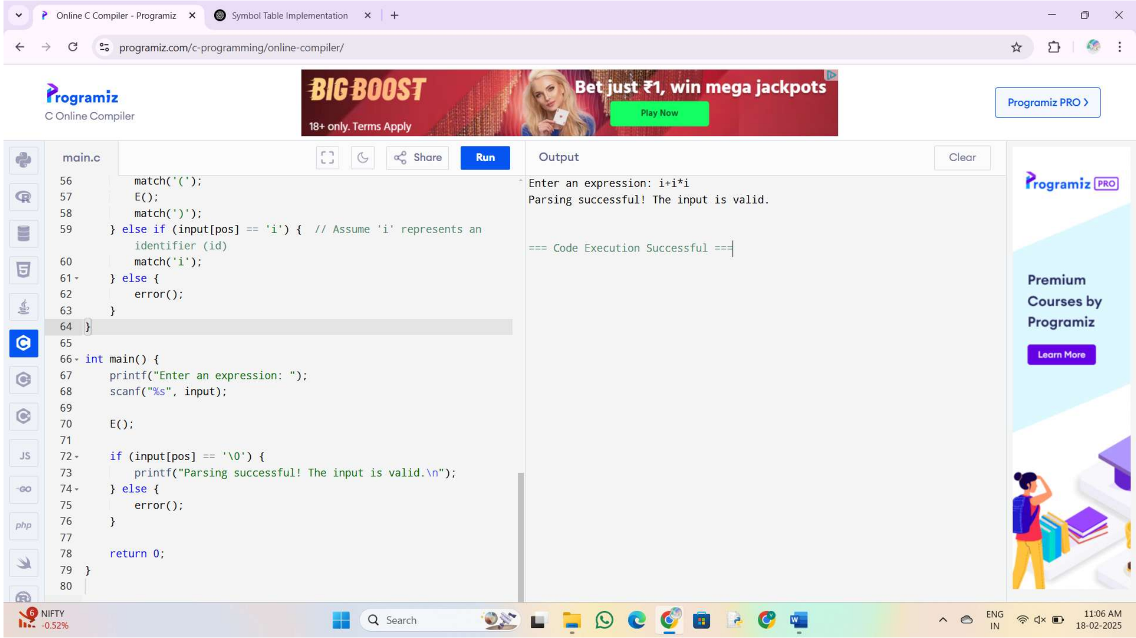
void T();

void Tprime();

void F();

```
void error() {
    printf("Error: Invalid syntax!\n");
    exit(0);
}

void match(char expected) {
    if (input[pos] == expected) {
        pos++;
    } else {
        error();
    }
}

void E() {
    T();
    Eprime();
}

void Eprime() {
    if (input[pos] == '+') {
        match('+');
        T();
        Eprime();
    }
}

void T() {
    F();
    Tprime();
}

void Tprime() {
    if (input[pos] == '*') {
        match('*');
        F();
        Tprime();
    }
}

void F() {
    if (input[pos] == '(') {
        match('(');
        E();
        match(')');
    } else if (input[pos] == 'i') {  // Assume 'i' represents an identifier (id)
        match('i');
    } else {
        error();
    }
}

int main() {
    printf("Enter an expression: ");
    scanf("%s", input);
    E();


    if (input[pos] == '\0') {
        printf("Parsing successful! The input is valid.\n");
    } else {
        error();
    }


    return 0;
}
```

Op:



Online C Compiler - Programiz
Symbol Table Implementation
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Programiz PRO >

main.c
Share
Run
Output
Clear

```
        match('(');
        E();
        match(')');
    } else if (input[pos] == 'i') {  // Assume 'i' represents an
        identifier (id)
        match('i');
    } else {
        error();
    }
}

int main() {
    printf("Enter an expression: ");
    scanf("%s", input);

    E();

    if (input[pos] == '\0') {
        printf("Parsing successful! The input is valid.\n");
    } else {
        error();
    }

    return 0;
}
```

```
Enter an expression: i+i*i
Parsing successful! The input is valid.


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

11.In a class of Grade 3, Mathematics Teacher asked for the Acronym PEMDAS?. All of them are thinking for a while. A smart kid of the class Kishore of the class says it is Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction. Can you write a C Program to help the students to understand about the operator precedence parsing for an expression containing more than one operator, the order of evaluation depends on the order of operations.

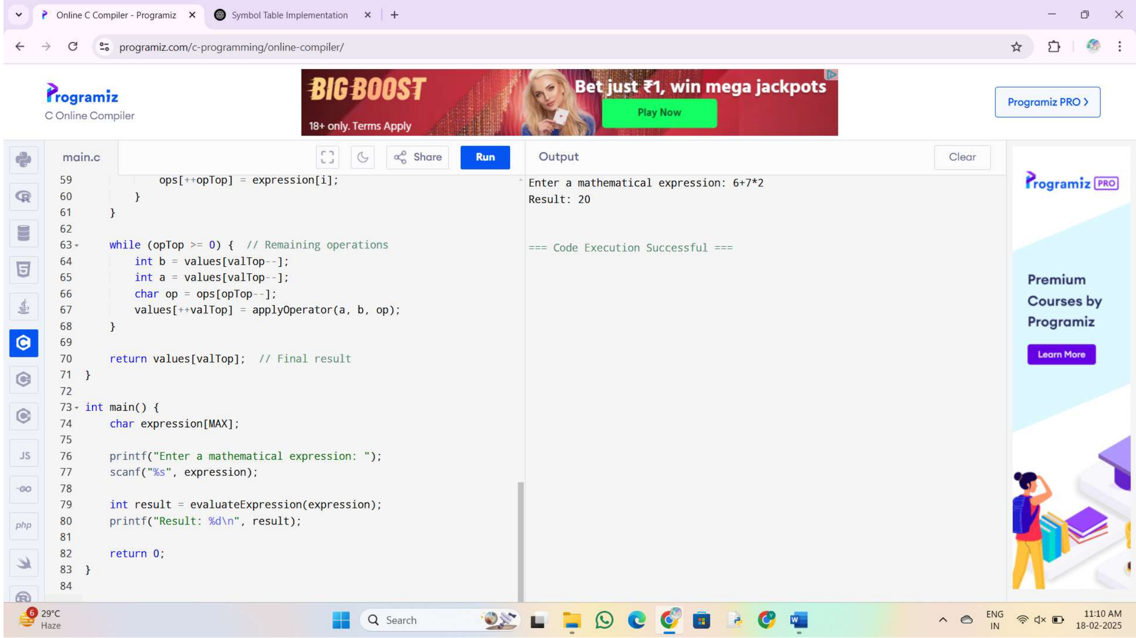
Code:

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

```
#include <stdlib.h>
#include <ctype.h>

#define MAX 100

int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 3;  // Exponentiation
    return 0;
}

int applyOperator(int a, int b, char op) {
    switch (op) {
        case '+': return a + b;
        case '-': return a - b;
        case '*': return a * b;
        case '/': return b ? a / b : 0;  // Avoid division by zero
        case '^': {
            int result = 1;
            for (int i = 0; i < b; i++) result *= a;
            return result;
        }
    }
    return 0;
}

int evaluateExpression(char* expression) {
```

```
int values[MAX], valTop = -1;   // Stack for numbers

char ops[MAX]; int opTop = -1;  // Stack for operators


for (int i = 0; expression[i] != '\0'; i++) {

    if (isdigit(expression[i])) {  // If number, push to value stack

        int num = 0;

        while (isdigit(expression[i])) {

            num = num * 10 + (expression[i] - '0');

            i++;

        }

        i--;  // Adjust index

        values[++valTop] = num;

    } else if (expression[i] == '(') {  // Left parenthesis

        ops[++opTop] = expression[i];

    } else if (expression[i] == ')') {  // Right parenthesis

        while (opTop >= 0 && ops[opTop] != '(') {

            int b = values[valTop--];

            int a = values[valTop--];

            char op = ops[opTop--];

            values[++valTop] = applyOperator(a, b, op);

        }

        opTop--;  // Pop '('

    } else {  // Operator

        while (opTop >= 0 && precedence(ops[opTop]) >= precedence(expression[i])) {

            int b = values[valTop--];

            int a = values[valTop--];

            char op = ops[opTop--];

            values[++valTop] = applyOperator(a, b, op);
            }
            ops[++opTop] = expression[i];
        }
    }

    while (opTop >= 0) {  // Remaining operations
        int b = values[valTop--];
        int a = values[valTop--];
        char op = ops[opTop--];
        values[++valTop] = applyOperator(a, b, op);
    }

    return values[valTop];  // Final result
}

int main() {
    char expression[MAX];

    printf("Enter a mathematical expression: ");
    scanf("%s", expression);

    int result = evaluateExpression(expression);
    printf("Result: %d\n", result);

    return 0;
}
```

Op:


Online C Compiler - Programiz
Symbol Table Implementation
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Programiz PRO
main.c
Share
Run
Output
Clear
Premium Courses by Programiz
Learn More
29°C
Haze
Search
ENG
IN
11:10 AM
18-02-2025

```
                ops[++opTop] = expression[i];
            }
        }

        while (opTop >= 0) {  // Remaining operations
            int b = values[valTop--];
            int a = values[valTop--];
            char op = ops[opTop--];
            values[++valTop] = applyOperator(a, b, op);
        }

        return values[valTop];  // Final result
    }

    int main() {
        char expression[MAX];

        printf("Enter a mathematical expression: ");
        scanf("%s", expression);

        int result = evaluateExpression(expression);
        printf("Result: %d\n", result);

        return 0;
    }
```

```
Enter a mathematical expression: 6+7*2
Result: 20

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

12.The main function of the Intermediate code generation is producing three address code statements for a given input expression. The three address codes help in determining the sequence in which operations are actioned by the compiler. The key work of Intermediate code generators is to simplify the process of Code Generator. Write a C Program to Generate the Three address code representation for the given input statement.

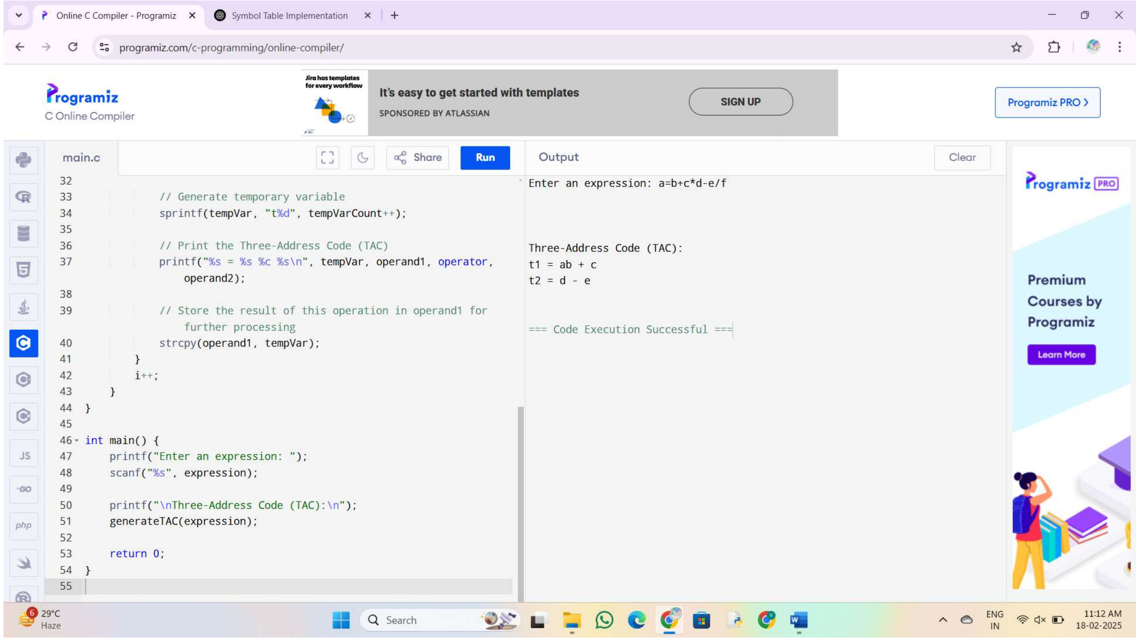
Code: #include <stdio.h>

```
#include <string.h>

#include <ctype.h>


#define MAX 100


char expression[MAX];

int tempVarCount = 1;


void generateTAC(char *exp) {

    char operand1[10], operand2[10], operator;
```

```
int i = 0, j = 0, k = 0;
char tempVar[5];

while (exp[i] != '\0') {
    if (isalnum(exp[i])) {  // If operand (variable or number)
        operand1[j++] = exp[i];
    } else if (strchr("+-*/", exp[i])) {  // If operator
        operand1[j] = '\0';
        j = 0;
        operator = exp[i];
        i++;

        while (exp[i] == ' ') i++;  // Ignore spaces

        while (isalnum(exp[i])) {  // Read second operand
            operand2[j++] = exp[i];
            i++;
        }
        operand2[j] = '\0';
        j = 0;

        // Generate temporary variable
        sprintf(tempVar, "t%d", tempVarCount++);

        // Print the Three-Address Code (TAC)
        printf("%s = %s %c %s\n", tempVar, operand1, operator, operand2);

        // Store the result of this operation in operand1 for further processing
```

```
            strcpy(operand1, tempVar);
        }
        i++;
    }
}

int main() {
    printf("Enter an expression: ");
    scanf("%s", expression);

    printf("\nThree-Address Code (TAC):\n");
    generateTAC(expression);

    return 0;
}
```

Op:



Online C Compiler - Programiz
Symbol Table Implementation
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Programiz PRO
main.c
Share
Run
Output
Clear
Programiz PRO
Premium Courses by Programiz
Learn More
29°C
Haze
Search
ENG
IN
11:12 AM
18-02-2025

```

            // Generate temporary variable
            sprintf(tempVar, "t%d", tempVarCount++);

            // Print the Three-Address Code (TAC)
            printf("%s = %s %c %s\n", tempVar, operand1, operator,
                operand2);

            // Store the result of this operation in operand1 for
                further processing
            strcpy(operand1, tempVar);
        }
        i++;
    }
}

int main() {
    printf("Enter an expression: ");
    scanf("%s", expression);

    printf("\nThree-Address Code (TAC):\n");
    generateTAC(expression);

    return 0;
}

```

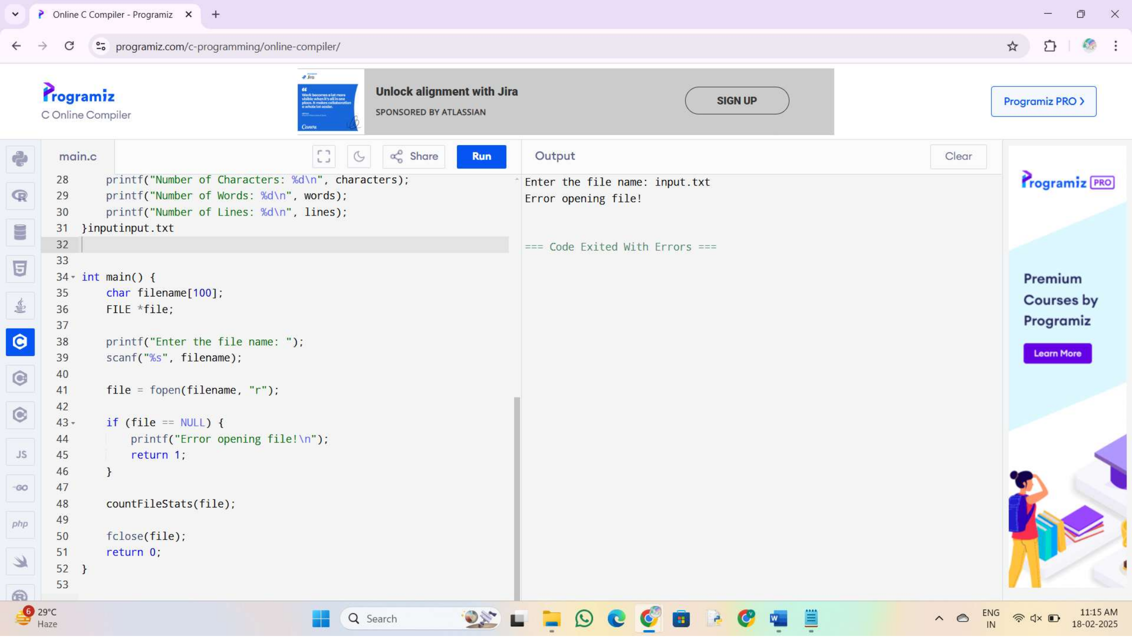
```
Enter an expression: a=b+c*d-e/f


Three-Address Code (TAC):
t1 = ab + c
t2 = d - e


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

13.Write a C program for implementing a Lexical Analyzer to Count the number of characters, words, and lines .

Code:

```
#include <stdio.h>
#include <stdlib.h>

void countFileStats(FILE *file) {
    int characters = 0, words = 0, lines = 0;
    char ch, prev = '\0';

    while ((ch = fgetc(file)) != EOF) {
        characters++;

        if (ch == '\n') {
            lines++;
        }

        // Check for word transition (space, newline, or EOF)
        if ((ch == ' ' || ch == '\n' || ch == '\t' || ch == EOF) && (prev != ' ' && prev != '\n' && prev != '\t')) {
            words++;
        }

        prev = ch;
    }

    // Handle last word if file doesn't end with a space or newline
    if (prev != ' ' && prev != '\n' && prev != '\t' && characters > 0) {
        words++;
    }

    printf("Number of Characters: %d\n", characters);
    printf("Number of Words: %d\n", words);
    printf("Number of Lines: %d\n", lines);
}inputinput.txt


int main() {
    char filename[100];
    FILE *file;

    printf("Enter the file name: ");
    scanf("%s", filename);

    file = fopen(filename, "r");

    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    countFileStats(file);

    fclose(file);
    return 0;
}
```

Op:



Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Programiz PRO >
main.c
Share
Run
Output
Clear

```c
    printf("Number of Characters: %d\n", characters);
    printf("Number of Words: %d\n", words);
    printf("Number of Lines: %d\n", lines);
}inputinput.txt


int main() {
    char filename[100];
    FILE *file;

    printf("Enter the file name: ");
    scanf("%s", filename);

    file = fopen(filename, "r");

    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    countFileStats(file);

    fclose(file);
    return 0;
}
```

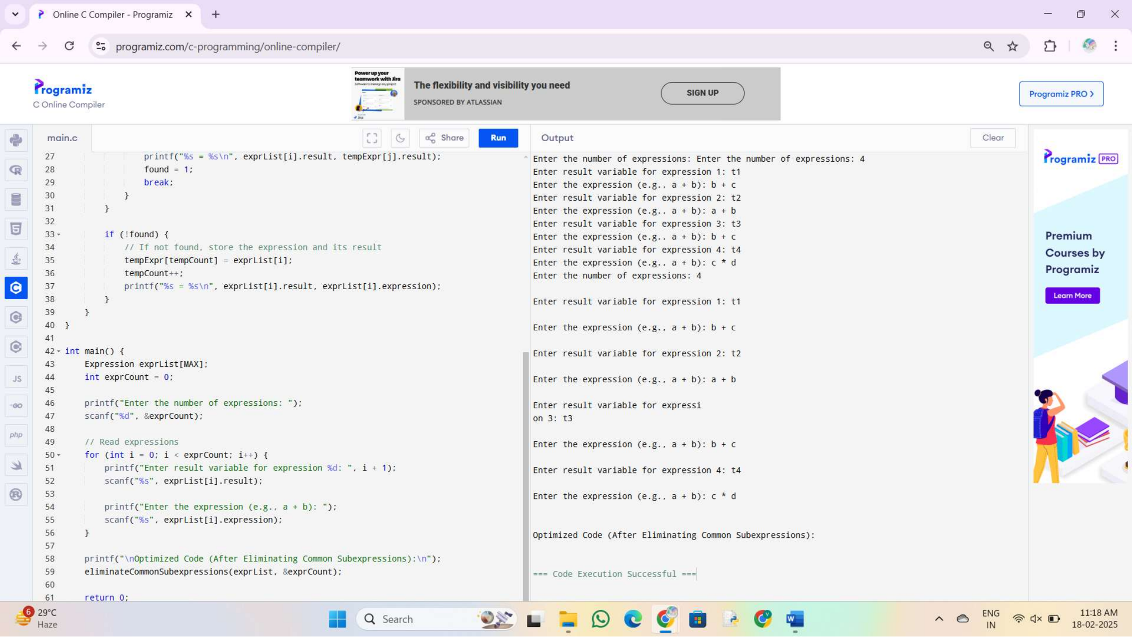
```
Enter the file name: input.txt
Error opening file!


=== Code Exited With Errors ===
```

14. Write a C Program for code optimization to eliminate common subexpression.

```c
#include <stdio.h>

#include <string.h>



#define MAX 100



// Structure to hold expression and its result variable

typedef struct {

    char expression[MAX];

    char result[MAX];

} Expression;



// Function to check if two expressions are identical

int areExpressionsEqual(char expr1[], char expr2[]) {

    return (strcmp(expr1, expr2) == 0);

}
```

```
// Function to optimize and eliminate common subexpressions
void eliminateCommonSubexpressions(Expression exprList[], int *exprCount) {
    Expression tempExpr[MAX];
    int tempCount = 0;

    for (int i = 0; i < *exprCount; i++) {
        int found = 0;
        for (int j = 0; j < tempCount; j++) {
            if (areExpressionsEqual(exprList[i].expression, tempExpr[j].expression)) {
                // If common subexpression found, replace it with previous result
                printf("%s = %s\n", exprList[i].result, tempExpr[j].result);
                found = 1;
                break;
            }
        }

        if (!found) {
            // If not found, store the expression and its result
            tempExpr[tempCount] = exprList[i];
            tempCount++;
            printf("%s = %s\n", exprList[i].result, exprList[i].expression);
        }
    }
}

int main() {
    Expression exprList[MAX];
    int exprCount = 0;
    printf("Enter the number of expressions: ");
    scanf("%d", &exprCount);

    // Read expressions
    for (int i = 0; i < exprCount; i++) {
        printf("Enter result variable for expression %d: ", i + 1);
        scanf("%s", exprList[i].result);

        printf("Enter the expression (e.g., a + b): ");
        scanf("%s", exprList[i].expression);
    }

    printf("\nOptimized Code (After Eliminating Common Subexpressions):\n");
    eliminateCommonSubexpressions(exprList, &exprCount);

    return 0;
}
```

op:


Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Programiz PRO
main.c
Share
Run
Output
Clear

```c
                printf("%s = %s\n", exprList[i].result, tempExpr[j].result);
                found = 1;
                break;
            }
        }

        if (!found) {
            // If not found, store the expression and its result
            tempExpr[tempCount] = exprList[i];
            tempCount++;
            printf("%s = %s\n", exprList[i].result, exprList[i].expression);
        }
    }
}

int main() {
    Expression exprList[MAX];
    int exprCount = 0;

    printf("Enter the number of expressions: ");
    scanf("%d", &exprCount);

    // Read expressions
    for (int i = 0; i < exprCount; i++) {
        printf("Enter result variable for expression %d: ", i + 1);
        scanf("%s", exprList[i].result);

        printf("Enter the expression (e.g., a + b): ");
        scanf("%s", exprList[i].expression);
    }

    printf("\nOptimized Code (After Eliminating Common Subexpressions):\n");
    eliminateCommonSubexpressions(exprList, &exprCount);

    return 0;
```

```
Enter the number of expressions: Enter the number of expressions: 4
Enter result variable for expression 1: t1
Enter the expression (e.g., a + b): b + c
Enter result variable for expression 2: t2
Enter the expression (e.g., a + b): a + b
Enter result variable for expression 3: t3
Enter the expression (e.g., a + b): b + c
Enter result variable for expression 4: t4
Enter the expression (e.g., a + b): c * d
Enter the number of expressions: 4

Enter result variable for expression 1: t1

Enter the expression (e.g., a + b): b + c

Enter result variable for expression 2: t2

Enter the expression (e.g., a + b): a + b

Enter result variable for expressi
on 3: t3

Enter the expression (e.g., a + b): b + c

Enter result variable for expression 4: t4

Enter the expression (e.g., a + b): c * d


Optimized Code (After Eliminating Common Subexpressions):


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

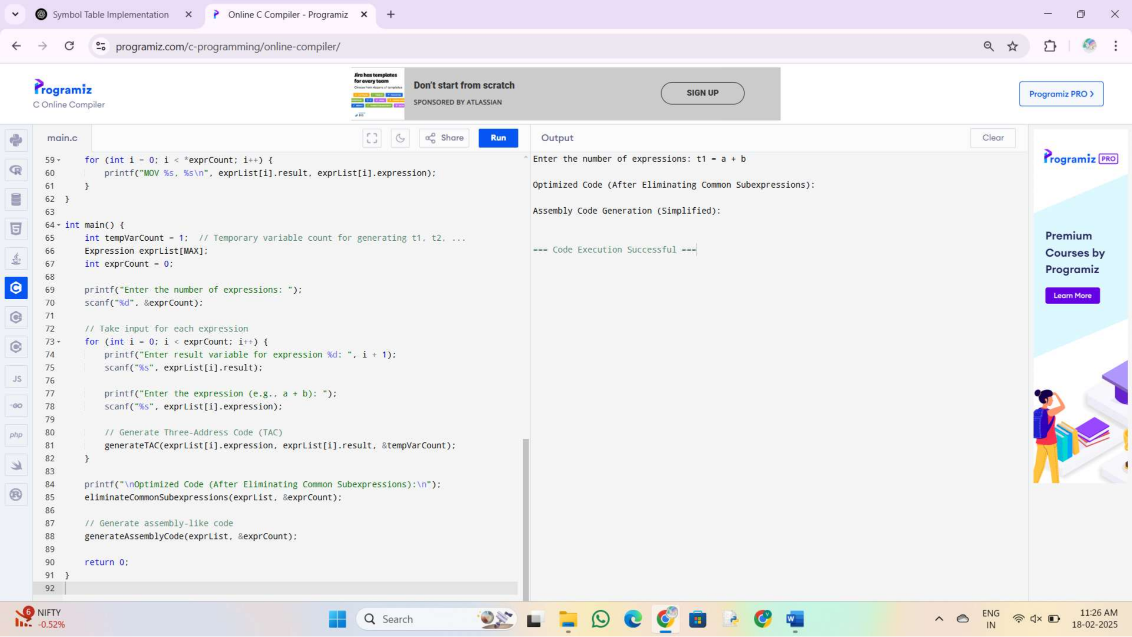
15.Write a C program to implement the back end of the compiler.

Code:

```c
#include <stdio.h>

#include <string.h>

#include <stdlib.h>



#define MAX 100



// Structure to hold an expression and its result

typedef struct {

    char expression[MAX];

    char result[MAX];

} Expression;



// Function to check if two expressions are identical

int areExpressionsEqual(char expr1[], char expr2[]) {
    return (strcmp(expr1, expr2) == 0);
}

// Function to generate Three-Address Code (TAC)
void generateTAC(char *expr, char *result, int *tempVarCount) {
    static char tempVar[MAX];
    sprintf(tempVar, "t%d", (*tempVarCount)++);  // Generate a temporary variable like t1,
t2...
    printf("%s = %s\n", tempVar, expr);  // Output the intermediate code (TAC)
    strcpy(result, tempVar);  // Store the result in the temporary variable
}

// Function to eliminate common subexpressions
void eliminateCommonSubexpressions(Expression exprList[], int *exprCount) {
    Expression tempExpr[MAX];
    int tempCount = 0;

    // Loop through all expressions
    for (int i = 0; i < *exprCount; i++) {
        int found = 0;

        // Check for common subexpression
        for (int j = 0; j < tempCount; j++) {
            if (areExpressionsEqual(exprList[i].expression, tempExpr[j].expression)) {
                // Reuse the result if expression is already computed
                printf("%s = %s\n", exprList[i].result, tempExpr[j].result);
                found = 1;
                break;
            }
        }

        // If no common subexpression, store it and output
        if (!found) {
            tempExpr[tempCount] = exprList[i];
            tempCount++;
            printf("%s = %s\n", exprList[i].result, exprList[i].expression);
        }
    }
}

// Function to simulate target code generation (assembly-like)
void generateAssemblyCode(Expression exprList[], int *exprCount) {
    printf("\nAssembly Code Generation (Simplified):\n");

    // Generating simplified assembly code (pseudo-code)
    for (int i = 0; i < *exprCount; i++) {
        printf("MOV %s, %s\n", exprList[i].result, exprList[i].expression);
    }
}

int main() {
    int tempVarCount = 1;  // Temporary variable count for generating t1, t2, ...
    Expression exprList[MAX];
    int exprCount = 0;

    printf("Enter the number of expressions: ");
```

```c
    scanf("%d", &exprCount);

    // Take input for each expression
    for (int i = 0; i < exprCount; i++) {
        printf("Enter result variable for expression %d: ", i + 1);
        scanf("%s", exprList[i].result);

        printf("Enter the expression (e.g., a + b): ");
        scanf("%s", exprList[i].expression);

        // Generate Three-Address Code (TAC)
        generateTAC(exprList[i].expression, exprList[i].result, &tempVarCount);
    }

    printf("\nOptimized Code (After Eliminating Common Subexpressions):\n");
    eliminateCommonSubexpressions(exprList, &exprCount);

    // Generate assembly-like code
    generateAssemblyCode(exprList, &exprCount);

    return 0;
}
```

Op:



```
    for (int i = 0; i < *exprCount; i++) {
        printf("MOV %s, %s\n", exprList[i].result, exprList[i].expression);
    }
}

int main() {
    int tempVarCount = 1;  // Temporary variable count for generating t1, t2, ...
    Expression exprList[MAX];
    int exprCount = 0;

    printf("Enter the number of expressions: ");
    scanf("%d", &exprCount);

    // Take input for each expression
    for (int i = 0; i < exprCount; i++) {
        printf("Enter result variable for expression %d: ", i + 1);
        scanf("%s", exprList[i].result);

        printf("Enter the expression (e.g., a + b): ");
        scanf("%s", exprList[i].expression);

        // Generate Three-Address Code (TAC)
        generateTAC(exprList[i].expression, exprList[i].result, &tempVarCount);
    }

    printf("\nOptimized Code (After Eliminating Common Subexpressions):\n");
    eliminateCommonSubexpressions(exprList, &exprCount);

    // Generate assembly-like code
    generateAssemblyCode(exprList, &exprCount);

    return 0;
}
```

```
Enter the number of expressions: t1 = a + b

Optimized Code (After Eliminating Common Subexpressions):

Assembly Code Generation (Simplified):


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

16.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. Write a LEX specification file to take input C program from a .c file and count t he number of characters, number of lines & number of words.

Input Source Program: (sample.c)

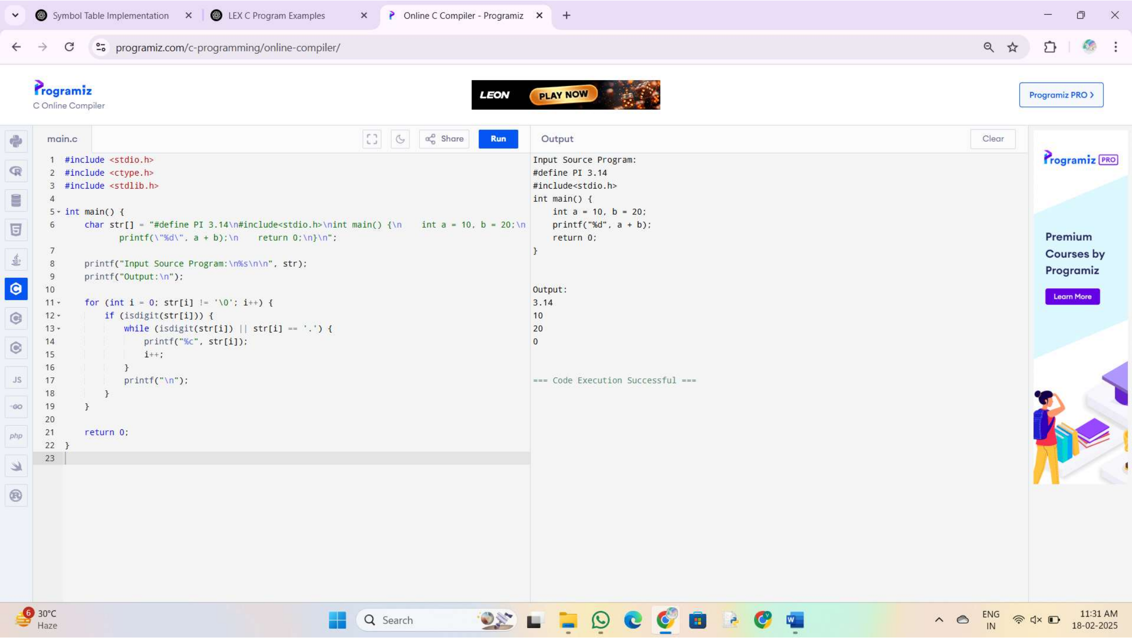
```
#include <stdio.h>

int main()

{

    int number1, number2, sum;

printf("Enter two integers: ");

scanf("%d %d", &number1, &number2);

sum = number1 + number2;

    printf("%d + %d = %d", number1, number2, sum);

 return 0;

}
```

Code:

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

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

int main() {
    char str[] = "#define PI 3.14\n#include<stdio.h>\nint main() {\n    int a = 10, b = 20;\n    printf(\"%d\", a + b);\n    return 0;\n}\n";

    printf("Input Source Program:\n%s\n\n", str);
    printf("Output:\n");

    for (int i = 0; str[i] != '\0'; i++) {
        if (isdigit(str[i])) {
            while (isdigit(str[i]) || str[i] == '.') {
                printf("%c", str[i]);
                i++;
            }
            printf("\n");
        }
    }

    return 0;
}
```

Op:



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

int main() {
    char str[] = "#define PI 3.14\n#include<stdio.h>\nint main() {\n    int a = 10, b = 20;\n
         printf(\"%d\", a + b);\n    return 0;\n}\n";

    printf("Input Source Program:\n%s\n\n", str);
    printf("Output:\n");

    for (int i = 0; str[i] != '\0'; i++) {
        if (isdigit(str[i])) {
            while (isdigit(str[i]) || str[i] == '.') {
                printf("%c", str[i]);
                i++;
            }
            printf("\n");
        }
    }

    return 0;
}
```

```
Input Source Program:
#define PI 3.14
#include<stdio.h>
int main() {
    int a = 10, b = 20;
    printf("%d", a + b);
    return 0;
}


Output:
3.14
10
20
0


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

17.Write a LEX program to print all the constants in the given C source program file.

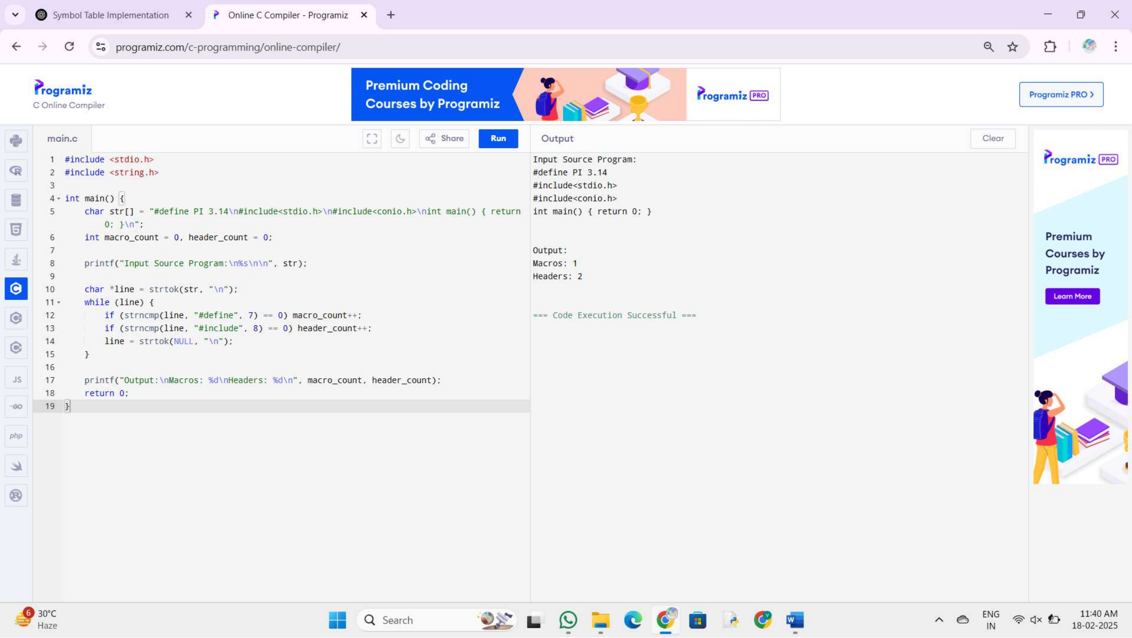
Input Source Program: (sample.c)

```
#define PI 3.14

#include<stdio.h> #include<conio.h>

 void main()

{

        int a,b,c = 30;

printf("hello");

}
```

Code:

```
#include <stdio.h>

#include <string.h>


int main() {

    char str[] = "#define PI 3.14\n#include<stdio.h>\n#include<conio.h>\nint main() { return 0; }\n";

    int macro_count = 0, header_count = 0;
    printf("Input Source Program:\n%s\n\n", str);


    char *line = strtok(str, "\n");
    while (line) {
        if (strncmp(line, "#define", 7) == 0) macro_count++;
        if (strncmp(line, "#include", 8) == 0) header_count++;
        line = strtok(NULL, "\n");
    }


    printf("Output:\nMacros: %d\nHeaders: %d\n", macro_count, header_count);
    return 0;
}
```

Op:



Symbol Table Implementation
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
main.c
Share
Run
Output
Clear
Premium Courses by Programiz
Learn More
30°C
Haze
Search
ENG
IN
11:40 AM
18-02-2025

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

int main() {
    char str[] = "#define PI 3.14\n#include<stdio.h>\n#include<conio.h>\nint main() { return
        0; }\n";
    int macro_count = 0, header_count = 0;

    printf("Input Source Program:\n%s\n\n", str);

    char *line = strtok(str, "\n");
    while (line) {
        if (strncmp(line, "#define", 7) == 0) macro_count++;
        if (strncmp(line, "#include", 8) == 0) header_count++;
        line = strtok(NULL, "\n");
    }

    printf("Output:\nMacros: %d\nHeaders: %d\n", macro_count, header_count);
    return 0;
}
```

```
Input Source Program:
#define PI 3.14
#include<stdio.h>
#include<conio.h>
int main() { return 0; }


Output:
Macros: 1
Headers: 2


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

18.Write a LEX program to count the number of Macros defined and header files included in the C program.

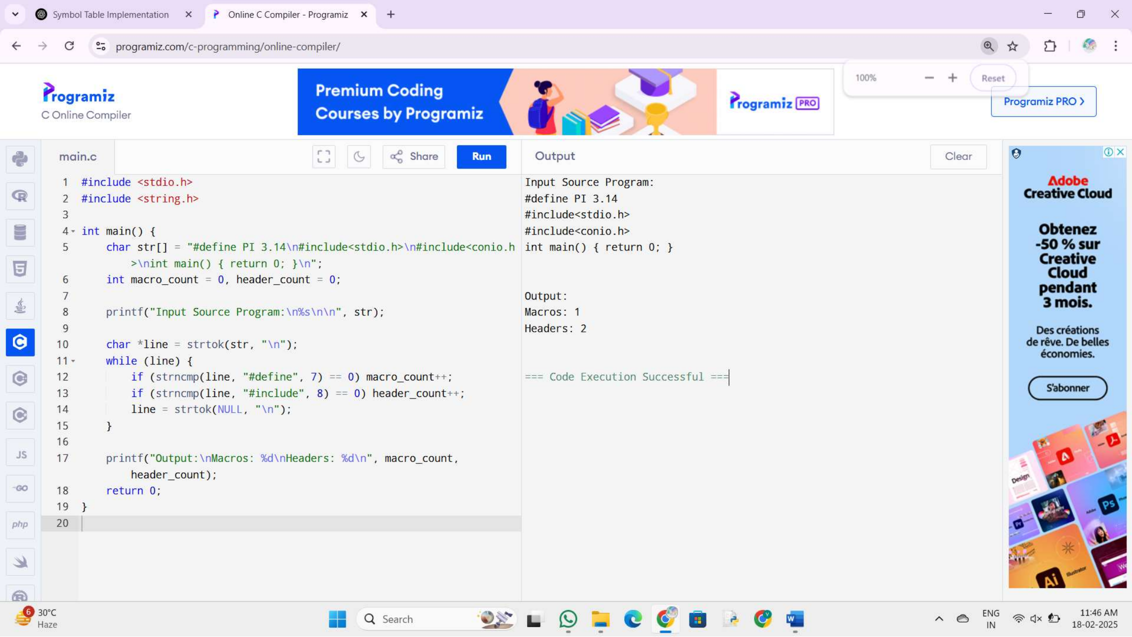
Input Source Program: (sample.c)

```
#define PI 3.14
#include<stdio.h>
#include<conio.h>
 void main()
{
int a,b,c = 30;
printf("hello");
}
```

Code:

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

int main() {
    char str[] = "#define PI 3.14\n#include<stdio.h>\n#include<conio.h>\nint main() { return 0; }\n";
    int macro_count = 0, header_count = 0;

    printf("Input Source Program:\n%s\n\n", str);

    char *line = strtok(str, "\n");
    while (line) {
        if (strncmp(line, "#define", 7) == 0) macro_count++;
        if (strncmp(line, "#include", 8) == 0) header_count++;
        line = strtok(NULL, "\n");
    }

    printf("Output:\nMacros: %d\nHeaders: %d\n", macro_count, header_count);
    return 0;
}
```

Op:



main.c

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

int main() {
    char str[] = "#define PI 3.14\n#include<stdio.h>\n#include<conio.h
        >\nint main() { return 0; }\n";
    int macro_count = 0, header_count = 0;

    printf("Input Source Program:\n%s\n\n", str);

    char *line = strtok(str, "\n");
    while (line) {
        if (strncmp(line, "#define", 7) == 0) macro_count++;
        if (strncmp(line, "#include", 8) == 0) header_count++;
        line = strtok(NULL, "\n");
    }

    printf("Output:\nMacros: %d\nHeaders: %d\n", macro_count,
        header_count);
    return 0;
}
```

Output

```
Input Source Program:
#define PI 3.14
#include<stdio.h>
#include<conio.h>
int main() { return 0; }


Output:
Macros: 1
Headers: 2


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

Symbol Table Implementation
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
100%
Reset
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Share
Run
Clear
30°C
Haze
Search
ENG
IN
11:46 AM
18-02-2025

19.Write a LEX program to print all HTML tags in the input file.

Input Source Program: (sample.html)

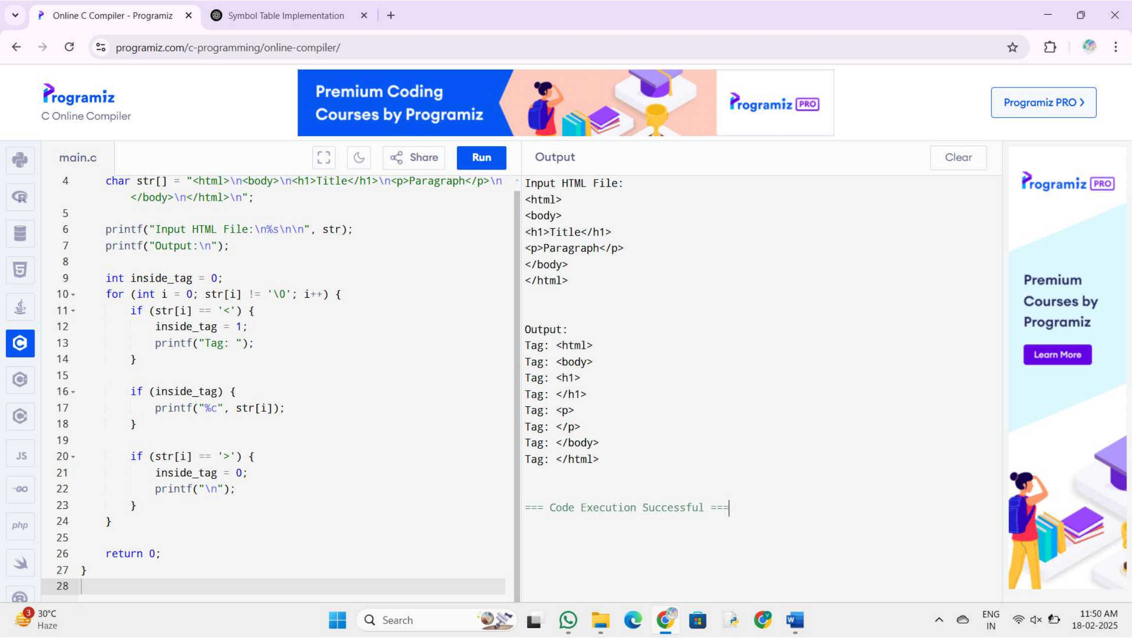
```
<html>

<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>

</html>
```

Code:

```
#include <stdio.h>


int main() {

    char str[] =
"<html>\n<body>\n<h1>Title</h1>\n<p>Paragraph</p>\n</body>\n</html>\n";
```

```
    printf("Input HTML File:\n%s\n\n", str);
    printf("Output:\n");

    int inside_tag = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == '<') {
            inside_tag = 1;
            printf("Tag: ");
        }

        if (inside_tag) {
            printf("%c", str[i]);
        }

        if (str[i] == '>') {
            inside_tag = 0;
            printf("\n");
        }
    }

    return 0;
}
```

Op:



Online C Compiler - Programiz
Symbol Table Implementation
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO >
main.c
Share
Run
Output
Clear
Premium Courses by Programiz
Learn More
30°C
Haze
Search
ENG
IN
11:50 AM
18-02-2025

```
    char str[] = "<html>\n<body>\n<h1>Title</h1>\n<p>Paragraph</p>\n
        </body>\n</html>\n";

    printf("Input HTML File:\n%s\n\n", str);
    printf("Output:\n");

    int inside_tag = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == '<') {
            inside_tag = 1;
            printf("Tag: ");
        }

        if (inside_tag) {
            printf("%c", str[i]);
        }

        if (str[i] == '>') {
            inside_tag = 0;
            printf("\n");
        }
    }

    return 0;
}
```

```
Input HTML File:
<html>
<body>
<h1>Title</h1>
<p>Paragraph</p>
</body>
</html>


Output:
Tag: <html>
Tag: <body>
Tag: <h1>
Tag: </h1>
Tag: <p>
Tag: </p>
Tag: </body>
Tag: </html>


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

20.Write a LEX program which adds line numbers to the given C program file and display the same in the standard output.

Input Source Program: (sample.c)

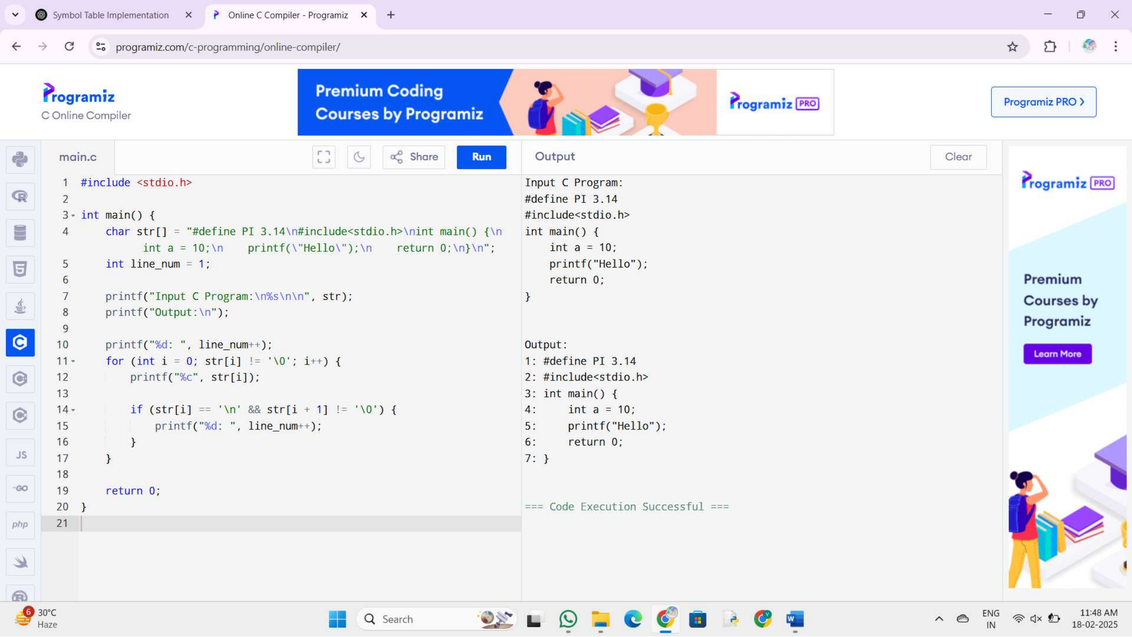
```
#define PI 3.14

#include<stdio.h>

#include<conio.h>

  void main()

{

int a,b,c = 30;

printf("hello");

}
```

Code:

```
#include <stdio.h>


int main() {

    char str[] = "#define PI 3.14\n#include<stdio.h>\nint main() {\n    int a = 10;\n
printf(\"Hello\");\n    return 0;\n}\n";
```

```
    int line_num = 1;


    printf("Input C Program:\n%s\n\n", str);
    printf("Output:\n");


    printf("%d: ", line_num++);
    for (int i = 0; str[i] != '\0'; i++) {
        printf("%c", str[i]);


        if (str[i] == '\n' && str[i + 1] != '\0') {
            printf("%d: ", line_num++);
        }
    }


    return 0;
}
```

Op:



Symbol Table Implementation
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Programiz PRO
main.c
Share
Run
Output
Clear

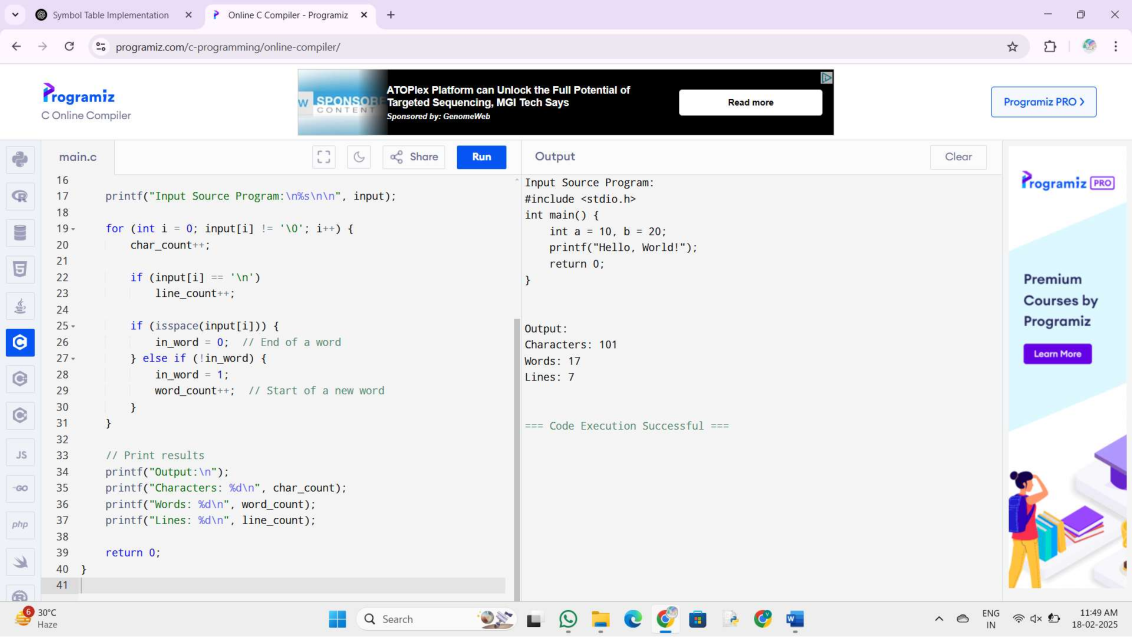
```
#include <stdio.h>

int main() {
    char str[] = "#define PI 3.14\n#include<stdio.h>\nint main() {\n
        int a = 10;\n    printf(\"Hello\");\n    return 0;\n}\n";
    int line_num = 1;

    printf("Input C Program:\n%s\n\n", str);
    printf("Output:\n");

    printf("%d: ", line_num++);
    for (int i = 0; str[i] != '\0'; i++) {
        printf("%c", str[i]);

        if (str[i] == '\n' && str[i + 1] != '\0') {
            printf("%d: ", line_num++);
        }
    }

    return 0;
}
```

```
Input C Program:
#define PI 3.14
#include<stdio.h>
int main() {
    int a = 10;
    printf("Hello");
    return 0;
}


Output:
1: #define PI 3.14
2: #include<stdio.h>
3: int main() {
4:     int a = 10;
5:     printf("Hello");
6:     return 0;
7: }


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

21. Write a LEX specification count the number of characters, number of lines & number of words.

```
#include <stdio.h>

#include <ctype.h>


int main() {

    // Sample Input: A small C program stored in a string

    char input[] =

        "#include <stdio.h>\n"

        "int main() {\n"

        "    int a = 10, b = 20;\n"

        "    printf(\"Hello, World!\");\n"

        "    return 0;\n"

        "}\n";


    int char_count = 0, word_count = 0, line_count = 1;

    int in_word = 0;


    printf("Input Source Program:\n%s\n\n", input);


    for (int i = 0; input[i] != '\0'; i++) {

        char_count++;


        if (input[i] == '\n')

            line_count++;


        if (isspace(input[i])) {

            in_word = 0;  // End of a word
        } else if (!in_word) {
            in_word = 1;
            word_count++;  // Start of a new word
        }
    }

    // Print results
    printf("Output:\n");
    printf("Characters: %d\n", char_count);
    printf("Words: %d\n", word_count);
    printf("Lines: %d\n", line_count);

    return 0;
}
```

Op:



Symbol Table Implementation
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Programiz PRO
main.c
Share
Run
Output
Clear
Premium Courses by Programiz
Learn More

```
    printf("Input Source Program:\n%s\n\n", input);

    for (int i = 0; input[i] != '\0'; i++) {
        char_count++;

        if (input[i] == '\n')
            line_count++;

        if (isspace(input[i])) {
            in_word = 0;  // End of a word
        } else if (!in_word) {
            in_word = 1;
            word_count++;  // Start of a new word
        }
    }

    // Print results
    printf("Output:\n");
    printf("Characters: %d\n", char_count);
    printf("Words: %d\n", word_count);
    printf("Lines: %d\n", line_count);

    return 0;
}
```

```
Input Source Program:
#include <stdio.h>
int main() {
    int a = 10, b = 20;
    printf("Hello, World!");
    return 0;
}


Output:
Characters: 101
Words: 17
Lines: 7


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