**1.Develop a lexical Analyzer to identify identifiers, constants, operators using C program.**

Program:

#include <stdio.h>

#include <ctype.h>

void lexicalAnalyzer(const char \*input) {

    int i = 0;

    while (input[i] != '\0') {

        if (isspace(input[i])) {

            i++;

        } else if (isalpha(input[i])) {

            printf("Identifier: %c\n", input[i]);

            i++;

        } else if (isdigit(input[i])) {

            printf("Constant: %c\n", input[i]);

            i++;

        } else {

            printf("Operator: %c\n", input[i]);

            i++;

        }

    }

}

int main() {

    char input[] = "a = 5 + b;";

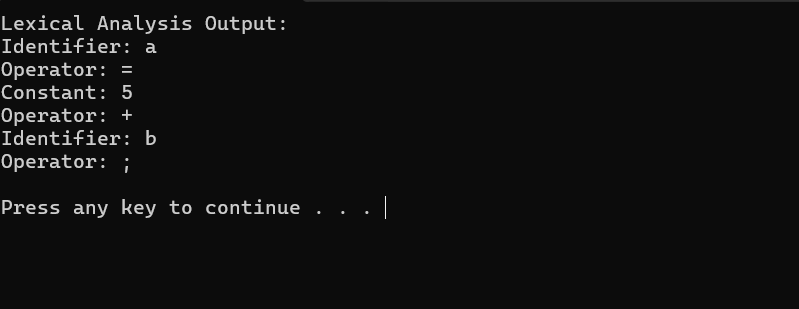
    printf("Lexical Analysis Output:\n");

    lexicalAnalyzer(input);

    return 0;

}

**Output:**



1. **Develop a lexical Analyzer to identify whether a given line is a comment or not using C.**

**Program:**

#include <stdio.h>

#include <string.h>

void checkComment(char \*line) {

    if (line[0] == '/' && line[1] == '/') {

        printf("Single-line Comment: %s\n", line);

    }

    else if (line[0] == '/' && line[1] == '\*') {

        printf("Multi-line Comment Start: %s\n", line);

        int len = strlen(line);

        if (len >= 4 && line[len - 2] == '\*' && line[len - 1] == '/') {

            printf("Multi-line Comment End: %s\n", line);

        }

    }

    else {

        printf("Not a comment: %s\n", line);

    }

}

int main() {

    char input[100];

    printf("Enter a line of code: ");

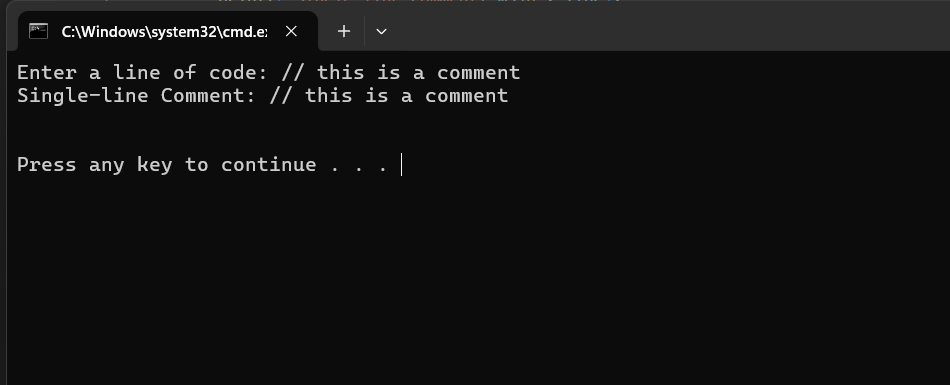
    fgets(input, sizeof(input), stdin);

    checkComment(input);

    return 0;

}

Output:



**Exp. No. 3**

**Design a lexical Analyzer for given language should ignore the redundant spaces, tabs and new lines and ignore comments using C**

**Program:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

void lexicalAnalyzer(const char \*input) {

    int i = 0;

    int in\_comment = 0;

    while (input[i] != '\0') {

        if (isspace(input[i])) {

            i++;

            continue;

        }

        if (input[i] == '/' && input[i + 1] == '/') {

            while (input[i] != '\0' && input[i] != '\n') {

                i++;

            }

            continue;

        }

        if (input[i] == '/' && input[i + 1] == '\*') {

            in\_comment = 1;

            i += 2;

            while (input[i] != '\0' && !(input[i] == '\*' && input[i + 1] == '/')) {

                i++;

            }

            if (input[i] != '\0') {

                i += 2;

            }

            in\_comment = 0;

            continue;

        }

        printf("%c", input[i]);

        i++;

    }

}

int main() {

    char input[] = "  int  x  =  10;  // This is a comment\n  /\* This is a \n   multi-line comment \*/ \n x = x + 5; ";

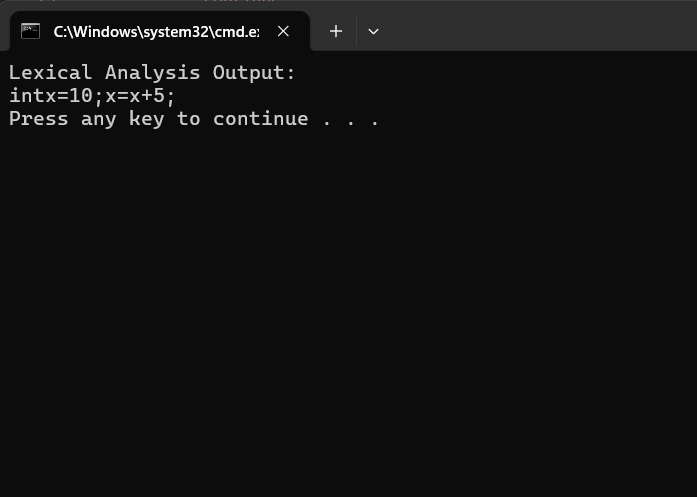
    printf("Lexical Analysis Output:\n");

    lexicalAnalyzer(input);

    return 0;

}

**Output;**



**Exp. No. 4**

**Design a lexical Analyzer to validate operators to recognize the operators +,-,\*,/ using regular arithmetic operators using C**

**Program:**

#include <stdio.h>

int isOperator(char ch) {

    return (ch == '+' || ch == '-' || ch == '\*' || ch == '/');

}

void lexicalAnalyzer(const char \*input) {

    int i = 0;

    while (input[i] != '\0') {

        if (isOperator(input[i])) {

            printf("Operator: %c\n", input[i]);

        }

        i++;

    }

}

int main() {

    char input[] = "a + b - c \* d / e";

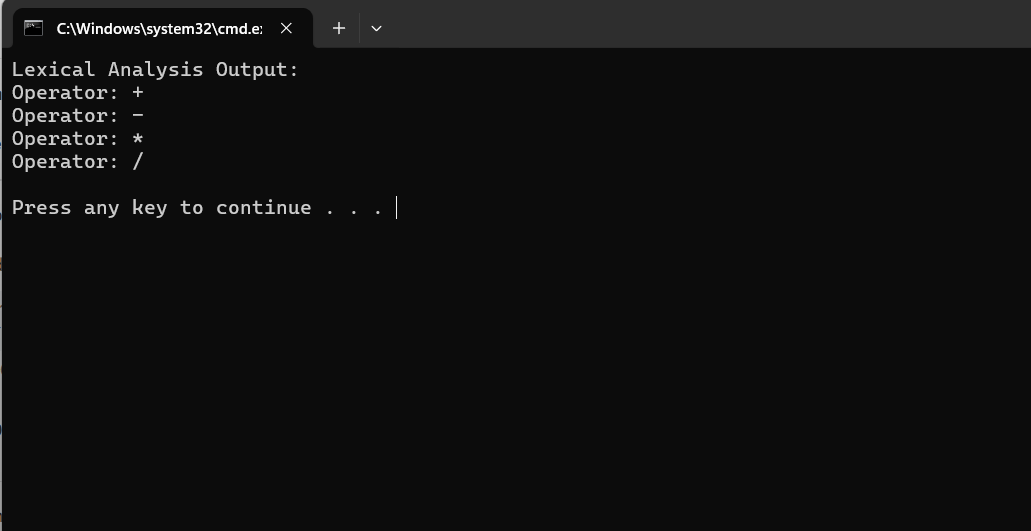
    printf("Lexical Analysis Output:\n");

    lexicalAnalyzer(input);

    return 0;

}

**Output:**



**Exp. No. 5**

**Design a lexical Analyzer to find the number of whitespaces and newline characters using C.**

**Program:**

#include <stdio.h>

void lexicalAnalyzer(const char \*input) {

    int spaces = 0, newlines = 0, i = 0;

    while (input[i] != '\0') {

        if (input[i] == ' ' || input[i] == '\t') {

            spaces++;

        } else if (input[i] == '\n') {

            newlines++;

        }

        i++;

    }

    printf("Number of Whitespaces: %d\n", spaces);

    printf("Number of Newlines: %d\n", newlines);

}

int main() {

    char input[] = "Hello World!  \nThis is a test.  \nAnother line.";

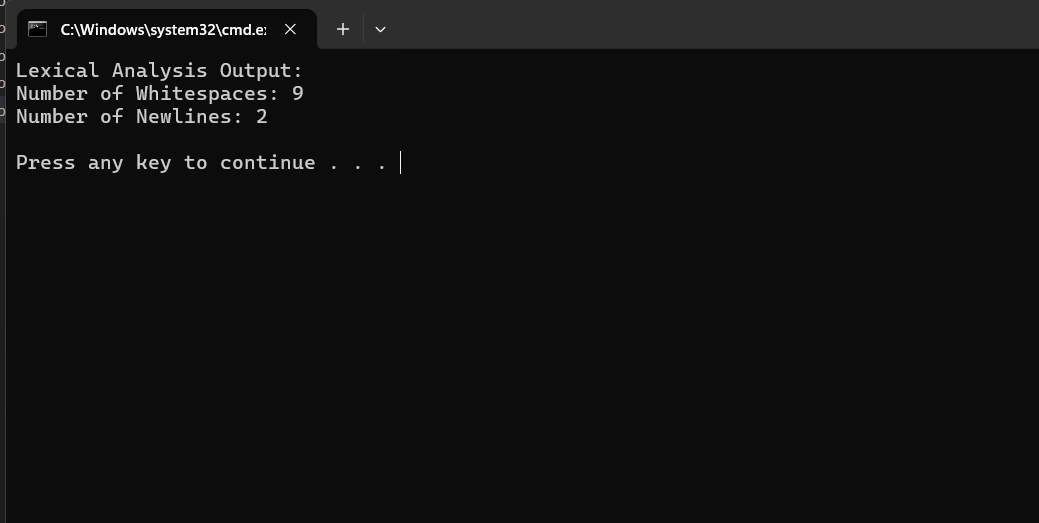
    printf("Lexical Analysis Output:\n");

    lexicalAnalyzer(input);

    return 0;

}

**Output;**



**Exp. No. 6**

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

**Program:**

#include <stdio.h>

#include <ctype.h>

int isValidIdentifier(const 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 input[100];

    printf("Enter an identifier: ");

    scanf("%s", input);

    if (isValidIdentifier(input))

        printf("Valid Identifier\n");

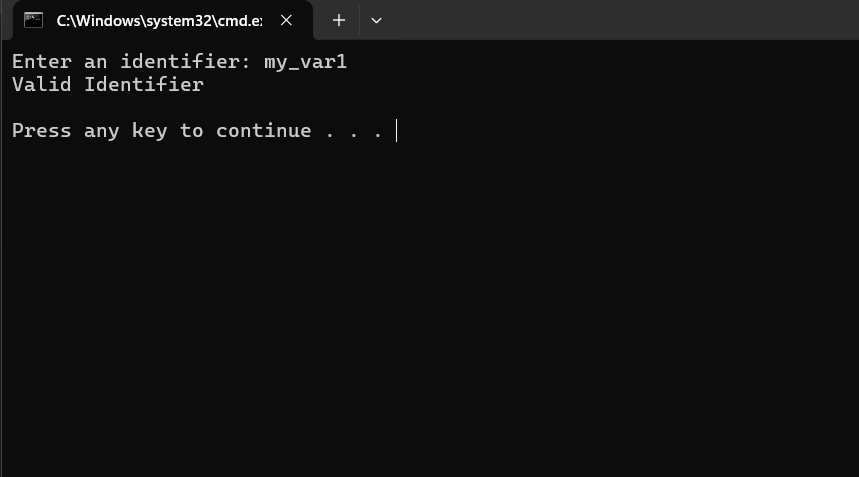
    else

        printf("Invalid Identifier\n");

    return 0;

}

**Output:**



**Exp. No. 8**

**Write a C program to find FOLLOW( ) - predictive parser for the given grammar.**

**Program;**