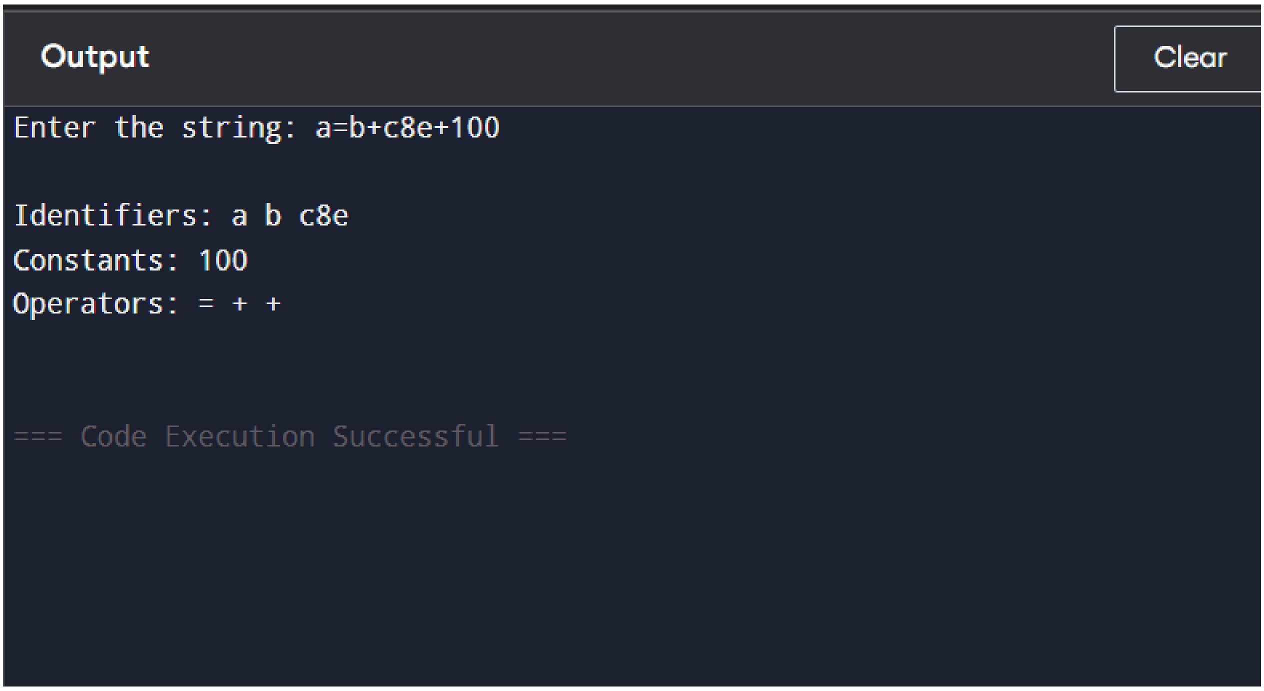
Experiment 1:

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.

Code:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int main() {
    char str[100];
    char identifiers[50][20], constants[50][20], operators[50];
    int i, j, idCount = 0, constCount = 0, opCount = 0;
    printf("Enter the string: ");
    fgets(str, sizeof(str), stdin);
    for (i = 0; str[i] != '\0'; i++) {
        if (isalpha(str[i]) || str[i] == '_') {
            j = 0;
            char temp[20];
            while (isalnum(str[i]) || str[i] == '_')
                temp[j++] = str[i++];
            temp[j] = '\0';
            strcpy(identifiers[idCount++], temp);
            i--;
        }
        else if (isdigit(str[i])) {
            j = 0;
            char temp[20];
            while (isdigit(str[i]))
                temp[j++] = str[i++];
            temp[j] = '\0';
            strcpy(constants[constCount++], temp);
            i--;
        }
        else if (str[i] == '+' || str[i] == '-' || str[i] == '*' || str[i] == '/' || str[i] == '=') {
            operators[opCount++] = str[i];
        }
    }
    printf("\nIdentifiers: ");
    for (i = 0; i < idCount; i++)
        printf("%s ", identifiers[i]);
    printf("\nConstants: ");
    for (i = 0; i < constCount; i++)
        printf("%s ", constants[i]);
    printf("\nOperators: ");
    for (i = 0; i < opCount; i++)
        printf("%c ", operators[i]);
    printf("\n");
    return 0;
}
```

Output:



```
Enter the string: a=b+c8e+100

Identifiers: a b c8e
Constants: 100
Operators: = + +



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

Experiment 2:

Extend the lexical Analyzer to Check comments, denied as follows in C:

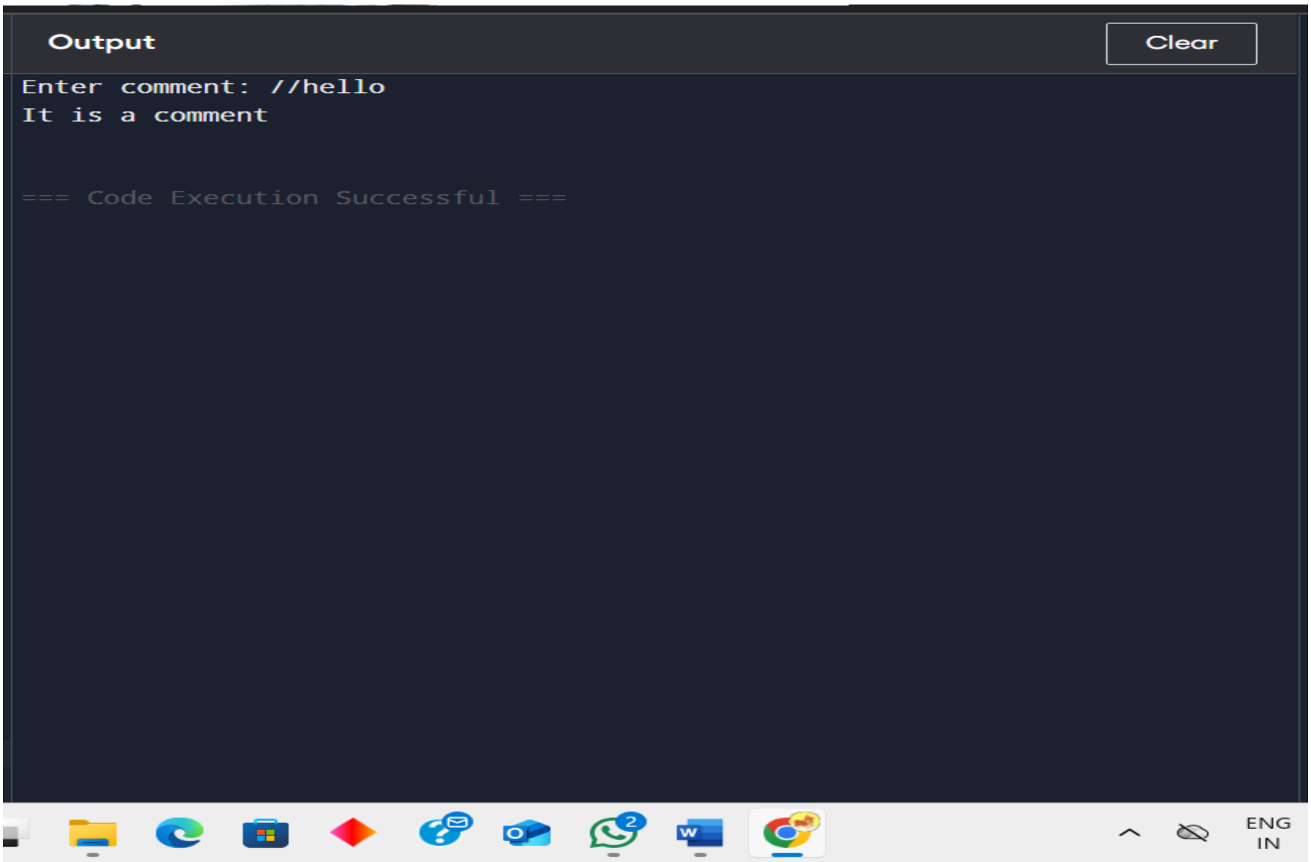
a) A comment begins with // and includes all characters until the end of that line.

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.

Code:

```
#include <stdio.h>
int main() {
    char com[100];
    int i, a = 0;
    printf("Enter comment: ");
    fgets(com, sizeof(com), stdin);
    if (com[0] == '/') {
        if (com[1] == '/')
            printf("It is a comment\n");
        else if (com[1] == '*') {
            for (i = 2; com[i] != '\0'; i++) {
                if (com[i] == '*' && com[i + 1] == '/') {
                    printf("It is a comment\n");
                    a = 1;
                    break;
                }
            }
            if (a == 0)
                printf("It is not a comment\n");
        } else
            printf("It is not a comment\n");
    } else
        printf("It is not a comment\n");
    return 0;
}
```

Output:



```
Enter comment: //hello
It is a comment


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

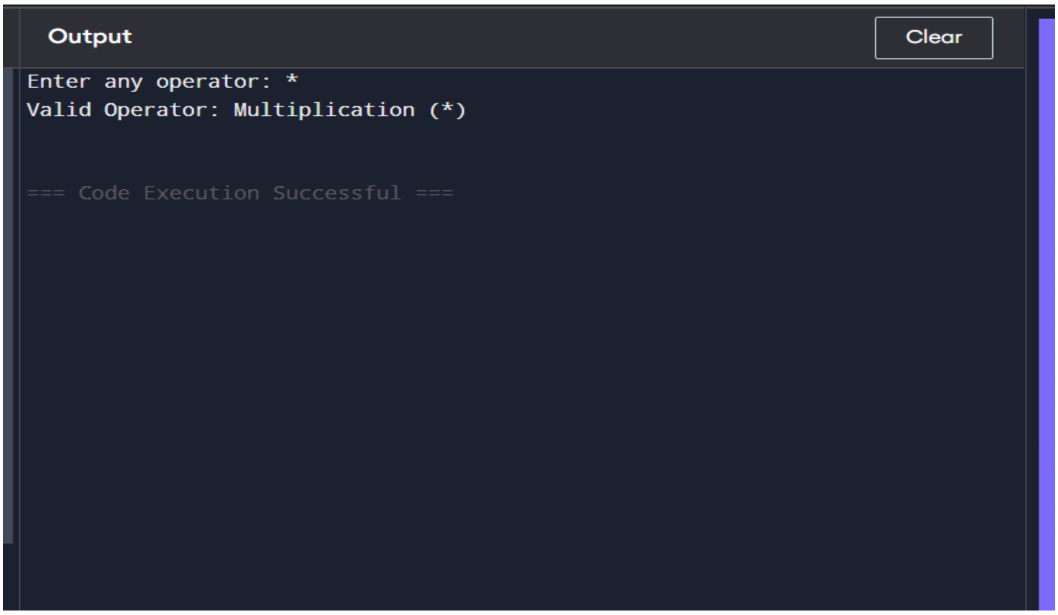
Experiment 3:

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

Code:

```
#include <stdio.h>
int main() {
    char op;
    printf("Enter any operator: ");
    scanf(" %c", &op);
    if (op == '+')
        printf("Valid Operator: Addition (+)\n");
    else if (op == '-')
        printf("Valid Operator: Subtraction (-)\n");
    else if (op == '*')
        printf("Valid Operator: Multiplication (*)\n");
    else if (op == '/')
        printf("Valid Operator: Division (/)\n");
    else
        printf("Invalid Operator!\n");
    return 0;
}
```

Output:



```
Enter any operator: *
Valid Operator: Multiplication (*)


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

Experiment 4:

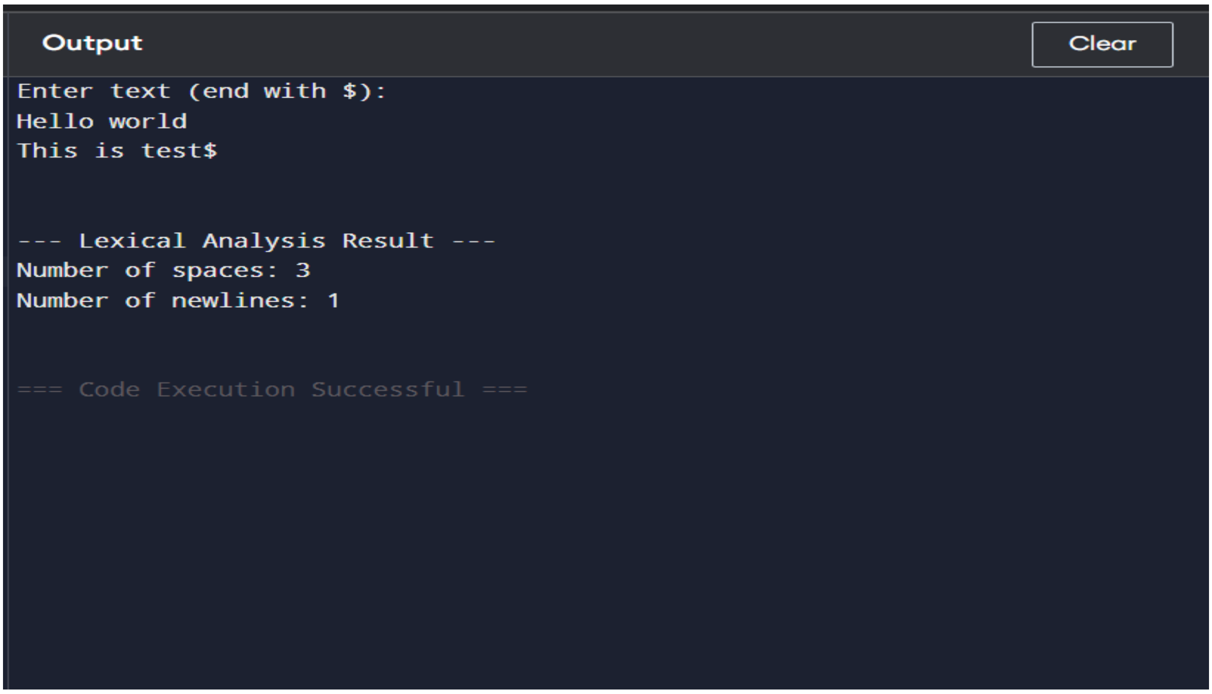
Design a lexical Analyzer to find the number of whitespaces and newline characters.

Code:

```
#include <stdio.h>
int main() {
    char ch;
    int spaces = 0, newlines = 0;
    printf("Enter text (end with $):\n");
    while ((ch = getchar()) != '$') {
        if (ch == ' ')
            spaces++;
        else if (ch == '\n')
            newlines++;
    }
    printf("\n--- Lexical Analysis Result ---\n");
    printf("Number of spaces: %d\n", spaces);
    printf("Number of newlines: %d\n", newlines);

    return 0;
}
```

Output:



```
Enter text (end with $):
Hello world
This is test$


--- Lexical Analysis Result ---
Number of spaces: 3
Number of newlines: 1


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

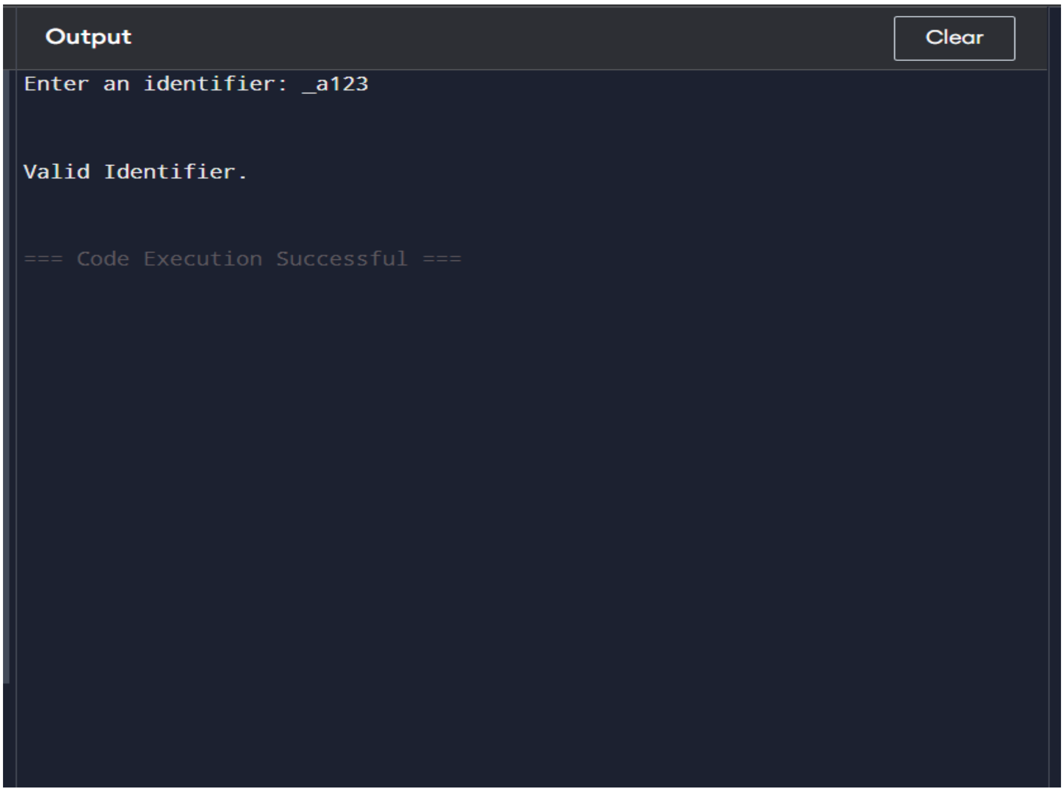
Experiment 5:

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

Code:

```
#include <stdio.h>
#include <ctype.h>
int main() {
    char id[50];
    int i, valid = 1;
    printf("Enter an identifier: ");
    scanf("%s", id);
    if (!isalpha(id[0]) && id[0] != '_')
        valid = 0;
    for (i = 1; id[i] != '\0'; i++) {
        if (!isalnum(id[i]) && id[i] != '_') {
            valid = 0;
            break;
        }
    }
    if (valid)
        printf("Valid Identifier.\n");
    else
        printf("Invalid Identifier.\n");
    return 0;
}
```

Output:



```
Enter an identifier: _a123


Valid Identifier.


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