**PROGRAM 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;

}

**LEX CODE:-**

%{

#include <stdio.h>

int char\_count = 0, word\_count = 0, line\_count = 0;

%}

%%

[ \t]+ { /\* Ignore spaces and tabs \*/ }

\n { line\_count++; }

"//".\* { /\* Ignore single-line comments \*/ }

"/\*"([^\*]|\\*+[^\*/])\*\\*+"/" { /\* Ignore multi-line comments \*/ }

[a-zA-Z\_][a-zA-Z0-9\_]\* { word\_count++; char\_count += yyleng; } /\* Identifiers \*/

[0-9]+ { word\_count++; char\_count += yyleng; } /\* Numbers \*/

. { char\_count++; } /\* Count all other characters \*/

%%

int main(int argc, char \*argv[]) {

if (argc != 2) {

printf("Usage: %s <input\_file.c>\n", argv[0]);

return 1;

}

FILE \*file = fopen(argv[1], "r");

if (!file) {

printf("Error: Could not open file %s\n", argv[1]);

return 1;

}

yyin = file;

yylex();

fclose(file);

printf("\nCharacter Count: %d\n", char\_count);

printf("Word Count: %d\n", word\_count);

printf("Line Count: %d\n", line\_count);

return 0;

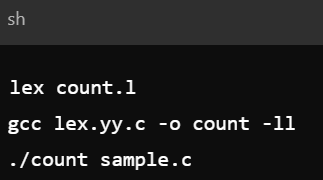
}

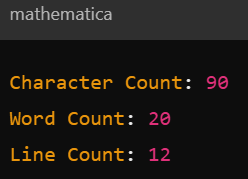
int yywrap() {

return 1;

}

**OUTPUT:-**

****

****

**C-CODE:-**

#include <stdio.h>

int main() {

int a = 5;

int b = 10;

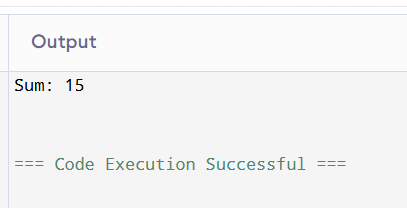
int sum = a + b;

printf("Sum: %d\n", sum);

return 0;

}

**OUTPUT:-**

****