Practical - 1

2CS701 – Compiler Construction

Harshit Gajipara

19bce059

**Aim:**

To implement lexical analyse to recognize all distinct token classes: use flex/lex tool to recognize all distinct token classes (Data type, Identifier, constant (Integer, Float, Char, String), Operator (Arithmetic, Relational, Assign, Unary +/-, Increment), Single line/Multi-line comments, Special symbol (;,{}())) .

Generate Lexical error reports for invalid lexeme.

**Code:**

%{

#include <stdio.h>

int tokens = 0;

int lines = 1;

%}

letter [a-zA-Z\_]

digit [0-9]

int\_literal -{digit}+|{digit}+

str\_literal \"[^\n].\*\"

keywords "for"|"NULL"|"struct"|"switch"|"continue"|"do"|"if"|"else"|"return"|"break"|"case"|"default"|"const"

keywords2 "auto"|"enum"|"extern"|"goto"|"register"|"short"|"signed"|"sizeof"|"static"|"switch"|"typedef"

keywords3 "union"|"unsigned"|"volatile"|"while"|"main"

datatype "int"|"float"|"char"|"double"|"void"|"long"

conditional\_operator ">="|"<="|"=="|">"|"<"|"!="

logical\_operator "||"|"&&"|"!"

bitwise\_operator "|"|"<<"|"~"|">>"|"^"

unary\_operator "++"|"--"

arithmatic\_operator "\*"|"+"|"/"|"-"

assignment\_operator "="|"\*="|"+="|"/="|"-="

identifier {letter}({letter}|{digit})\*

iofunction 'printf'|'scanf'

%%

\n {lines++; printf("\nLine %d\n",lines);}

[ \t] {}

[()#{}|;:&,\[\]] {tokens++; printf("SPECIAL\_SYMBOL\t\t:\t%s\n",yytext);}

{keywords}|{keywords2}|{keywords3} {tokens++; printf("KEYWORD\t\t:\t%s\n",yytext);}

{arithmatic\_operator} {tokens++; printf("ARITHMATIC\_OPERATOR\t:\t%s\n",yytext);}

{assignment\_operator} {tokens++; printf("ASSIGNMENT\_OPERATOR\t:\t%s\n",yytext);}

{conditional\_operator} {tokens++; printf("CONDITIONAL\_OPERATOR\t:\t%s\n",yytext);}

{unary\_operator} {tokens++; printf("UNARY\_OPERATOR\t:\t%s\n",yytext);}

{datatype} {tokens++; printf("DATA\_TYPE\t\t:\t%s\n",yytext);}

{identifier} {tokens++; printf("IDENTIFIER\t\t:\t%s\n",yytext);}

{int\_literal} {tokens++; printf("INT\_LITERAL\t\t:\t%s\n",yytext);}

{int\_literal}"."{int\_literal} {tokens++; printf("FLOAT\_LITERAL\t\t:\t%s\n",yytext);}

{str\_literal} {tokens++;printf("STR\_CONSTANT\t\t:\t%s\n",yytext);}

{iofunction} {tokens++; printf("IO\_FUNCTION\t\t:\t%s\n",yytext);}

\/\/[^\n].\* {printf("COMMENT\t\t:\t%s\n",yytext);} //single line comment

\/\\*[^(\*/)]\*\\*\/ {printf("COMMENT\t\t:\t%s\n",yytext);} //multi line comment

%%

int yywrap(){}

int main(){

printf("Line 1\n");

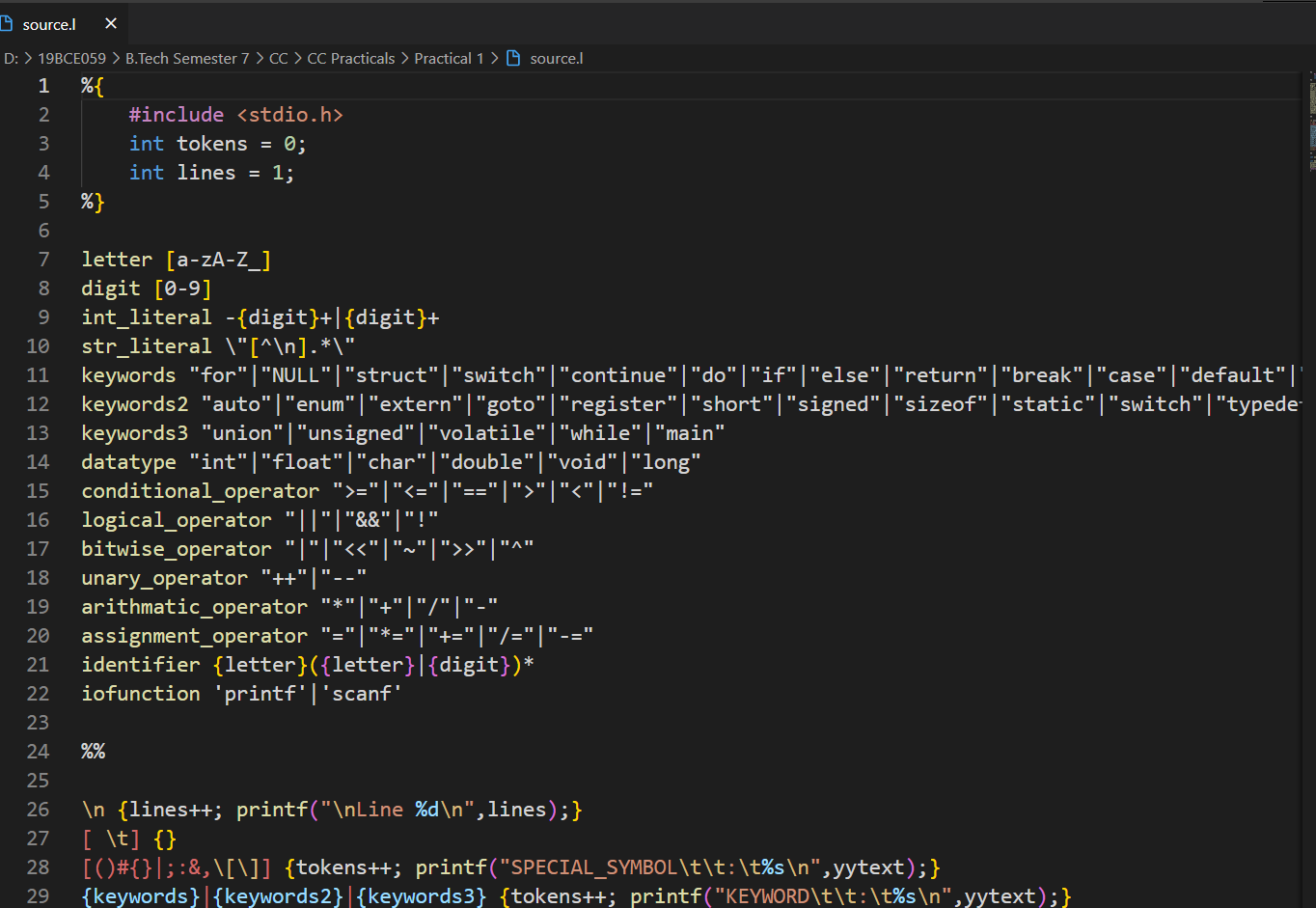
yylex();

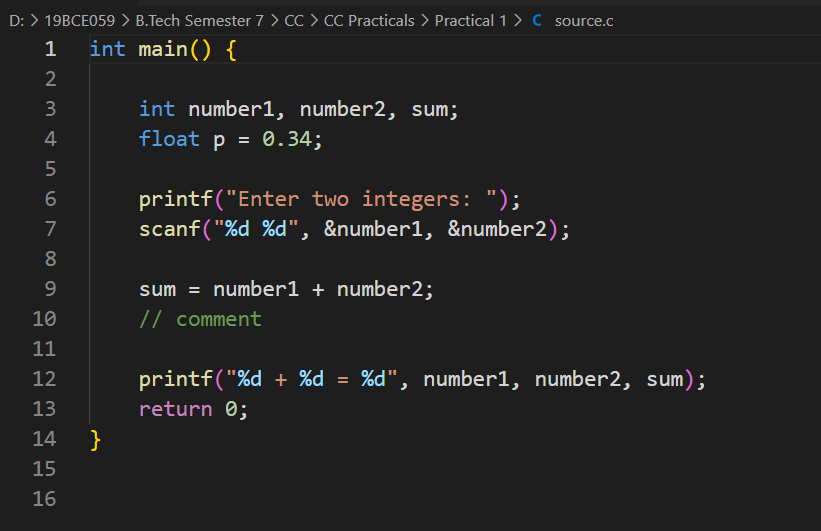
printf("\n\nTotal tokens = %d",tokens);

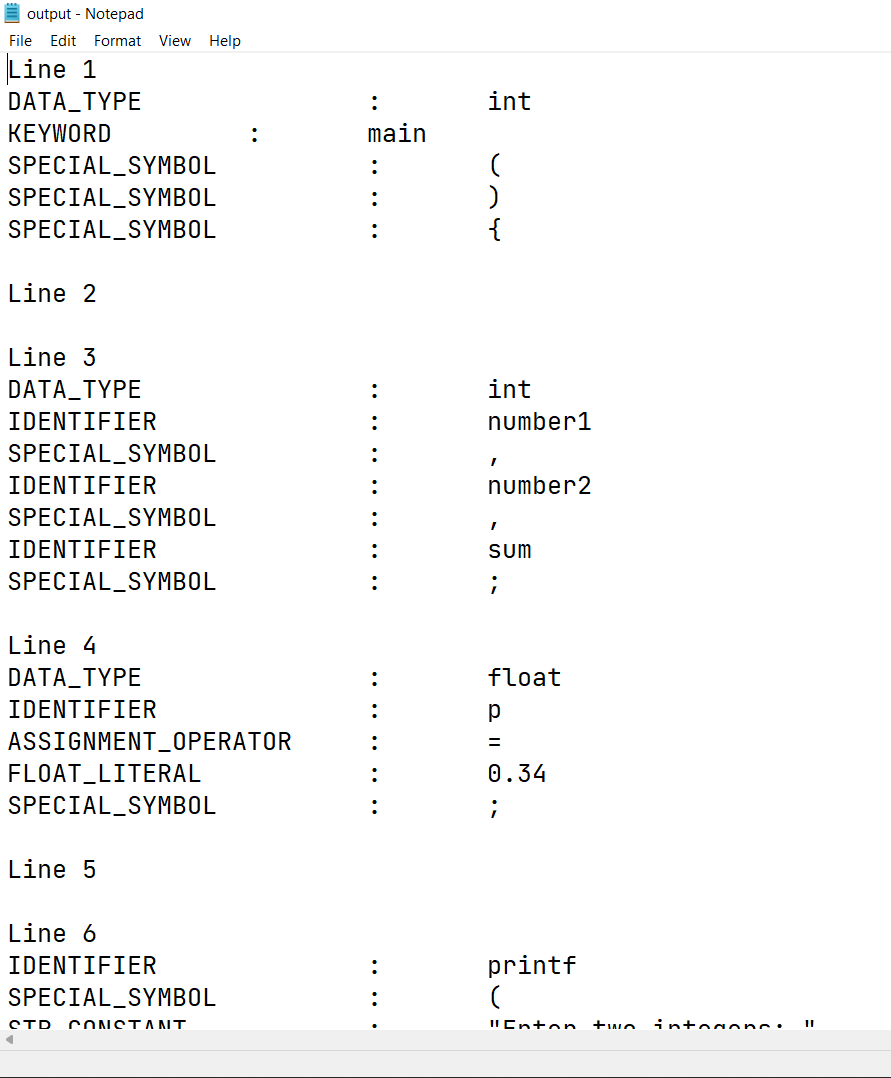
return 0;

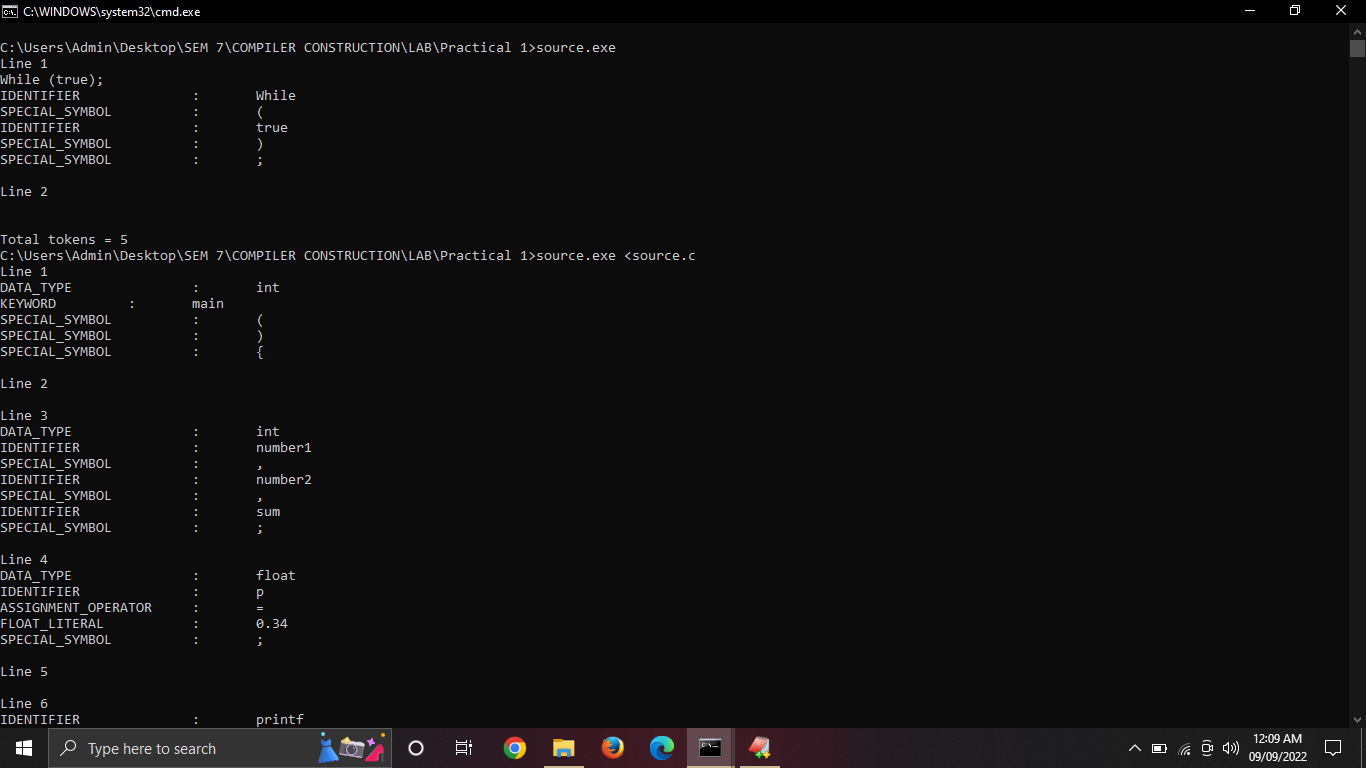
}

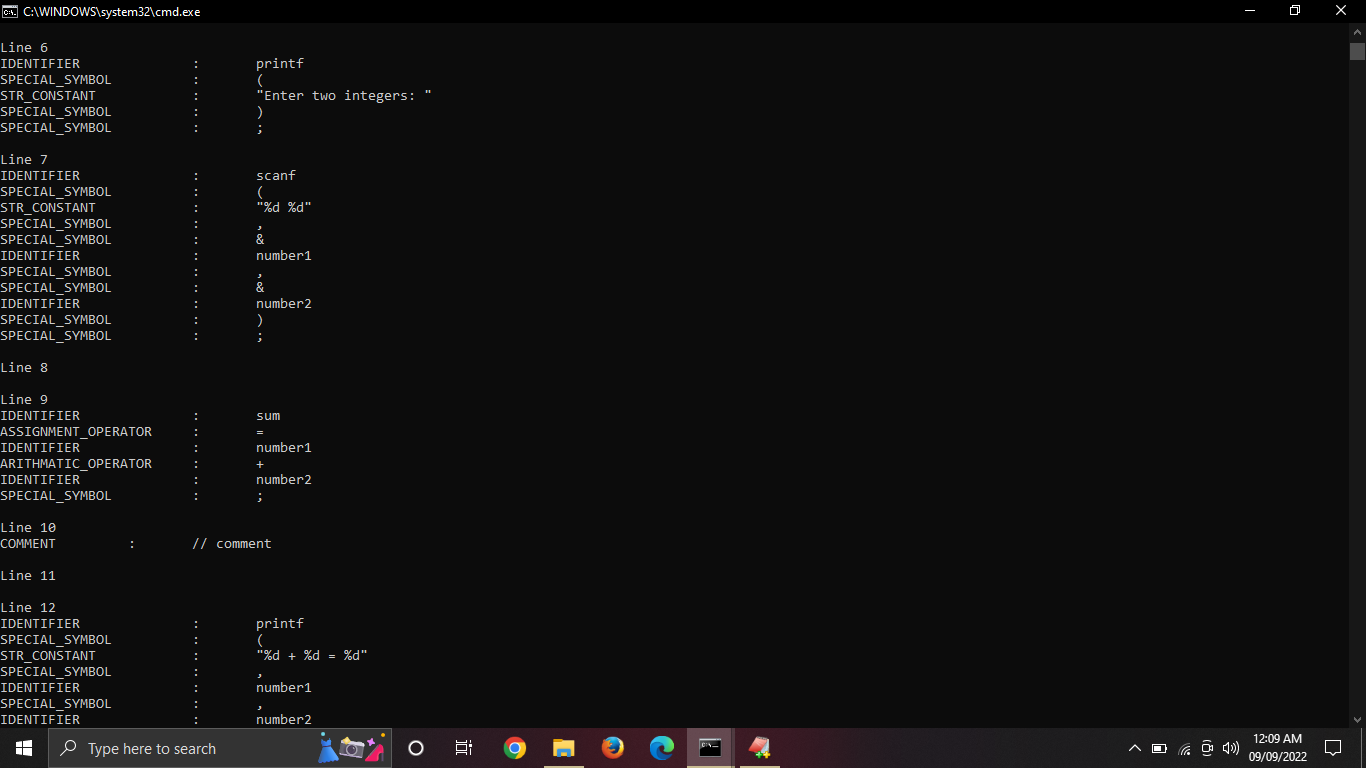
**Output:**

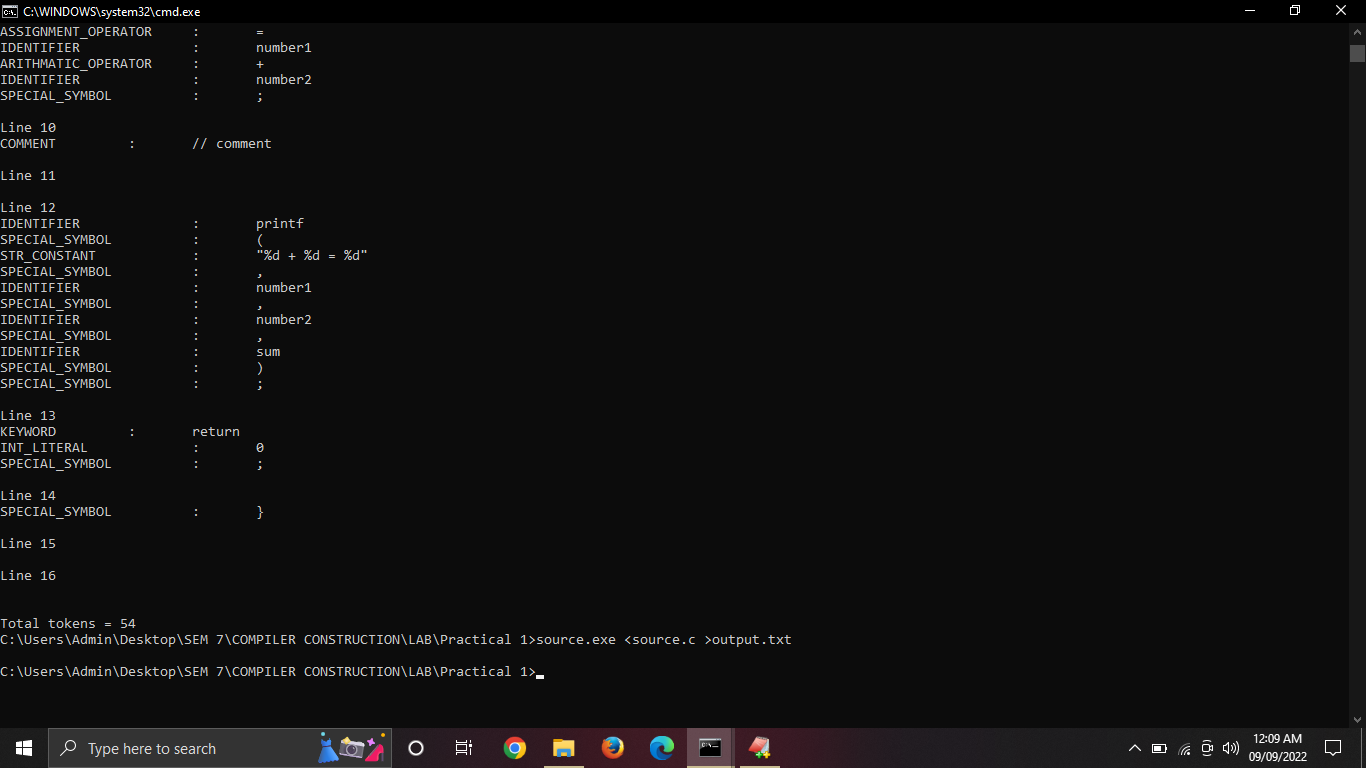












**Conclusion:**

In this practical, we learnt that using lex tool we can generate tokens from the source file and pass it to next level in compiler. Also we can design our own tokens and language based on it using lex.