# System Software (CS306)

## Assignment - 6

## U19CS012

1) Write a program to Implement Lexical Analyzer (**Lexer**).

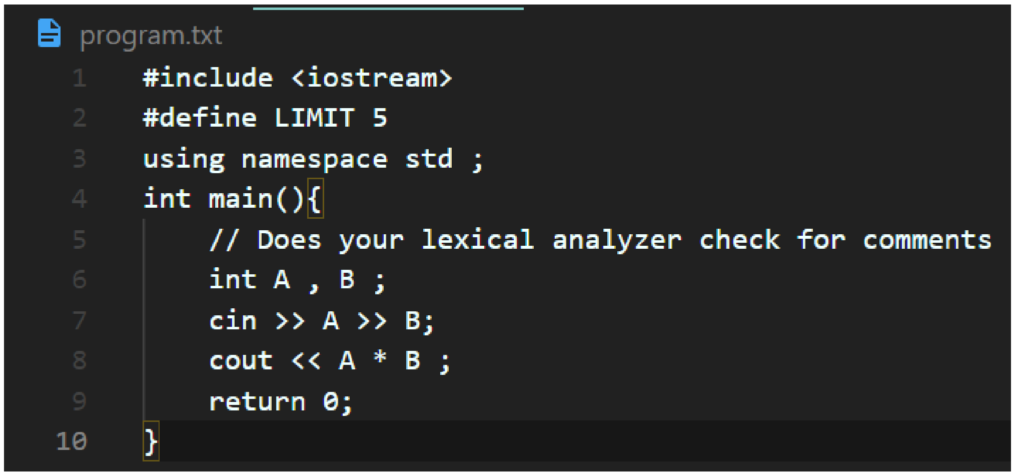
**Code:**

```
#include <bits/stdc++.h>
// For Regular Expression
#include <regex>
#include <time.h>
#include <iterator>
#include <windows.h>
#define deb(x) cout << #x << " = " << x << endl

using namespace std;

// This will Map the Regular Expression to Particular Category
map<string, string> Make_Regex_Map();

// Function to Classify the Tokens according to Different Category
map<size_t, pair<string, string>> Match_Language(map<string, string> patterns, string str);

// Function to Return the Operator's Category
string get_category(string op);

int main()
{
    ofstream fout;
    cout << endl
         << endl
         << endl;
    cout.fill(' ');
    cout.width(100);
    fout.open("OutputFile");
    char c;

    string filename;
    cout << "ENTER THE SOURCE CODE FILE NAME: Example \"abc.txt\" \n";
    cin >> filename;
    fstream fin(filename, fstream::in);

    string str;
    // Fetching Source Code in String type 'str'
    if (fin.is_open())
    {
        while (fin >> noskipws >> c)
            str = str + c;

        // Making a map which which will define the regex in source code to its pattern in my
language.
        map<string, string> patterns = Make_Regex_Map();

        /*DECLARING MAP 'lang_matches' from 'patterns' map which will pair up the patterns
        from the ['Source Code':'Defined Pattern' via a Regex named 'compare'. */
        map<size_t, pair<string, string>> lang_matches = Match_Language(patterns, str);

        // Writing matches in File ignoring 'spaces' and '\n'.
        int count = 1;
        cout << "\t\t\t\t--------------------------------------------------------------------
------------------------------ \n";
        cout.width(40);
        cout << "\t        NUMBER" << setw(10) << "              TOKEN "
             << " "
             << "            " << setw(20) << " PATTERN \n";
        cout.fill(' ');
        cout.width(40);

        cout << "\t\t\t\t--------------------------------------------------------------------
------------------------------ \n\n\n";

        // cout<<"\t\t\t\t                              PROCESSING SOURCE
CODE.......                                        \n\n\n";
        // Sleep(5000);

        for (auto match = lang_matches.begin(); match != lang_matches.end(); ++match)
        {
            // Not a Space or Comment, then Proceed
            if (!(match->second.first == " ") && !(match->second.first == "//"))
            {
                // Print if it is Variable or Identifier
                if (match->second.second == "Variable" || match->second.second ==
"Identifier")
                {
                    cout.width(40);
                    if (count < 10)
                    {
                        string double_digits = to_string(count);
                        double_digits = "0" + double_digits;
                        cout << "\t Token   No :" << double_digits << "  |   " << setw(10) <<
match->second.first << " "
                             << " -------->  |" << setw(25) << match->second.second <<
setw(18) << " ,  POINTER TO SYMBOL TABLE    " << endl;
                        fout << "\t Token   No :" << double_digits << "  |   " << setw(10) <<
match->second.first << " "
```

```
                             << " ------->  |" << setw(25) << match->second.second <<
setw(18) << " ,  POINTER TO SYMBOL TABLE    " << endl;
                        Sleep(1500);
                    }
                    else
                    {
                        cout << "\t Token   No :" << count << "  |   " << setw(10) << match-
>second.first << " "
                             << " ------->  |" << setw(25) << match->second.second <<
setw(18) << " ,  POINTER TO SYMBOL TABLE    " << endl;
                        fout << "\t Token   No :" << count << "  |   " << setw(10) << match-
>second.first << " "
                             << " ------->  |" << setw(25) << match->second.second <<
setw(18) << " ,  POINTER TO SYMBOL TABLE    " << endl;
                        Sleep(1500);
                    }
                    count++;
                }
                else
                {
                    // If Given Token is Operator
                    if (match->second.second == "Operator")
                    {
                        cout.width(40);
                        string op = get_category(match->second.first);
                        if (count < 10)
                        {
                            string double_digits = to_string(count);
                            double_digits = "0" + double_digits;
                            cout << "\t Token   No :" << double_digits << "  |   " <<
setw(10) << match->second.first << " "
                                 << " ------->  |" << setw(25) << match->second.second << " ,
" << op << "    " << endl;
                            fout << "\t Token   No :" << double_digits << "  |   " <<
setw(10) << match->second.first << " "
                                 << " ------->  |" << setw(25) << match->second.second << " ,
" << op << "    " << endl;
                            count++;
                        }
                        else
                        {
                            cout << "\t Token   No :" << count << "  |   " << setw(10) <<
match->second.first << " "
                                 << " ------->  |" << setw(25) << match->second.second << " ,
" << op << "    " << endl;
                            fout << "\t Token   No :" << count << "  |   " << setw(10) <<
match->second.first << " "
                                 << " ------->  |" << setw(25) << match->second.second << " ,
" << op << "    " << endl;
                            Sleep(1500);
                            count++;
                        }
                    }
                    else
                    {
                        cout.width(40);
                        if (count < 10)
                        {
                            string double_digits = to_string(count);
                            double_digits = "0" + double_digits;
                            cout << "\t Token   No :" << double_digits << "  |   " <<
setw(10) << match->second.first << " "
                                 << " ------->  |" << setw(25) << match->second.second <<
"    " << endl;
                            fout << "\t Token   No :" << double_digits << "  |   " <<
setw(10) << match->second.first << " "
                                 << " ------->  |" << setw(25) << match->second.second <<
"    " << endl;
                            count++;
                        }
                        else
                        {
                            cout << "\t Token   No :" << count << "  |   " << setw(10) <<
match->second.first << " "
                                 << " ------->  |" << setw(25) << match->second.second <<
"    " << endl;
                            fout << "\t Token   No :" << count << "  |   " << setw(10) <<
match->second.first << " "
                                 << " ------->  |" << setw(25) << match->second.second <<
"    " << endl;
                            count++;
                        }
                    }
                }
            }
        }

        string command = " ";

        while (command != "EXIT")
        {
            cout.fill(' ');
            cout.width(40);
            cout << "\n\n\t PRESS TYPE `EXIT` TO CLOSE WINDOW.\n\t NOTE: AN OUTPUT FILE WILL
BE GENERATED IN THE SAME FOLDER AS `Output.txt` \n";
            cin.width(40);
            cin >> command;

            if (command == "exit" || command == "EXIT" || command == "Exit")
                break;
            else
            {
                cout.fill(' ');
                cout.width(40);
                cout << "Please enter correct word.";
                cin.width(10);
                cin >> command;
            }
        }
    }

    else
    {
        cout.fill(' ');
        cout.width(40);
        cout << "\n FILE NOT FOUND!\n\n";
    }

    return 0;
}

// This will Map the Regular Expression to Particular Category
map<string, string> Make_Regex_Map()
{
    map<string, string> my_map{
        {"\\;|\\{|\\}|\\(|\\)|\\,|\\#", "Special Symbol"},
        {"auto|break|case|char|const|continue|default|do|double|else|enum|extern|float|for|go
to|if|int|long|register|return|short|signed|sizeof|switch|typedef|union|cin|cout|main|unsigne
d|void|volatile|while|using|namespace|std", "Keywords"},
        {"\\include|define|pragma|ifndef|endif", "Pre-Processor Directive"},
        {"\\iostream|\\stdio|\\string", "Library"},
        {"\\*|\\+|\\>>|\\<<|<|>", "Operator"},
        {"[0-9]+", "Integer"},
        {"[^include][^iostream][^int][^main][^cin][^cout][^;][^>>][^,][^[B ;cin]][a-z]+",
"Identifier"},
        {"[A-Z]+", "Variable"},
        {"[ ]", ""},
    };
    return my_map;
}

// Function to Classify the Tokens according to Different Category
map<size_t, pair<string, string>> Match_Language(map<string, string> patterns, string str)
{
    map<size_t, pair<string, string>> lang_matches;

    for (auto i = patterns.begin(); i != patterns.end(); ++i)
    {
        regex compare(i->first);
        auto words_begin = sregex_iterator(str.begin(), str.end(), compare);
        auto words_end = sregex_iterator();

        // MAKING PAIRS OF [STRING OF REGEX 'compare' : 'pattern']
        for (auto it = words_begin; it != words_end; ++it)
            lang_matches[it->position()] = make_pair(it->str(), i->second);
    }
    return lang_matches;
}

// Function to Return the Operator's Category
string get_category(string op)
{
    if (op == "*")
        return "MUL";
    else if (op == "+")
        return "ADD";
    else if (op == ">>")
        return "INS";
    else if (op == "<<")
        return "EXTR";
    else if (op == ">")
        return "RSHFT";
    else if (op == "<")
        return "LSHFT";
    else if (op == "/")
        return "DIV";
    else if (op == "%")
        return "MOD";
    else if (op == "++")
        return "INCREMENT";
    else if (op == "--")
        return "DECREMENT";
    else if (op == "=")
        return "ASSIGNMENT";
    else if (op == "?:")
        return "CONDITIONAL";
    else
        return "Special Op";
}
```

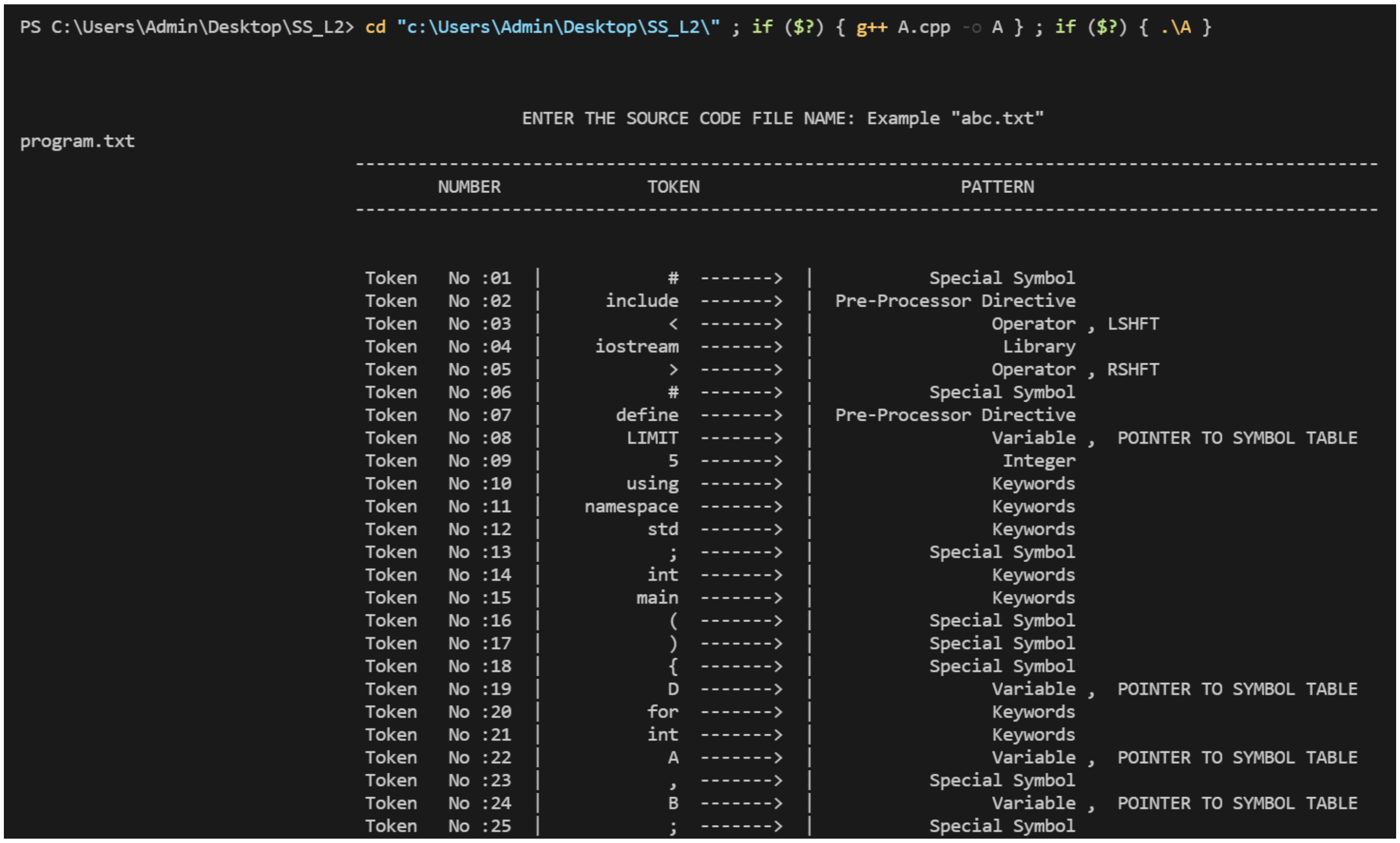
## **Test-Cases:**

Input File ("program.txt")

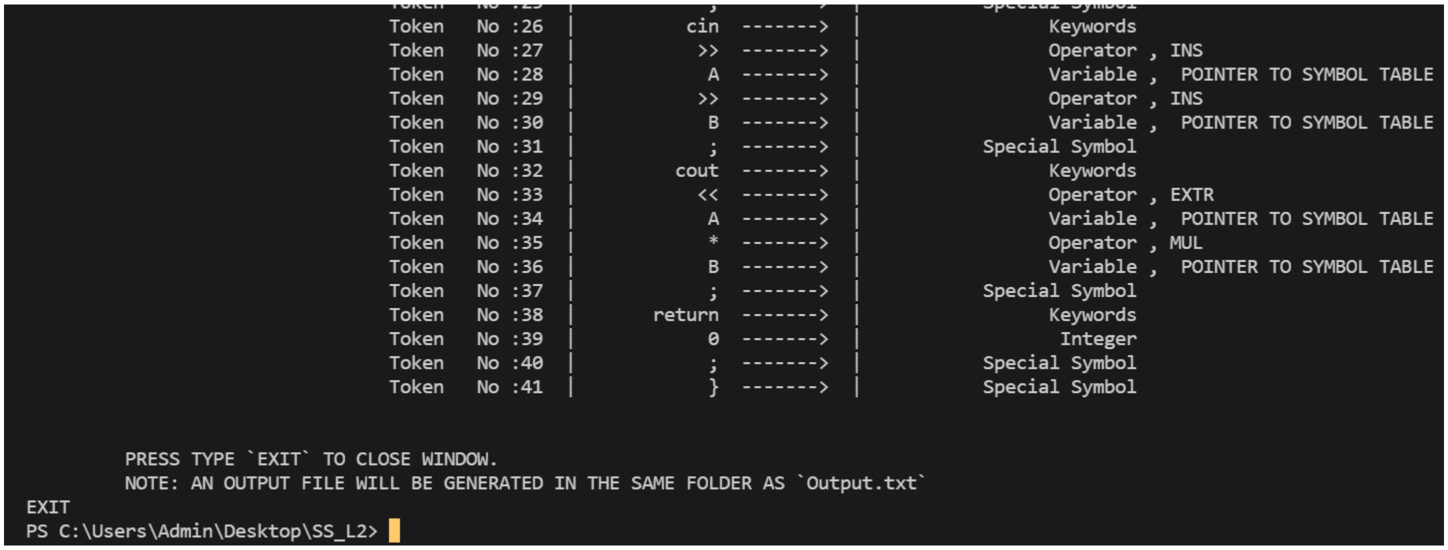


program.txt

```
#include <iostream>
#define LIMIT 5
using namespace std ;
int main(){
    // Does your lexical analyzer check for comments
    int A , B ;
    cin >> A >> B;
    cout << A * B ;
    return 0;
}
```



```
PS C:\Users\Admin\Desktop\SS_L2> cd "c:\Users\Admin\Desktop\SS_L2\" ; if ($?) { g++ A.cpp -o A } ; if ($?) { .\A }


                                    ENTER THE SOURCE CODE FILE NAME: Example "abc.txt"
program.txt
                        -------------------------------------------------------------------------------------------------
                              NUMBER             TOKEN                          PATTERN
                        -------------------------------------------------------------------------------------------------


                        Token   No :01  |          #  ------->  |          Special Symbol
                        Token   No :02  |    include  ------->  |  Pre-Processor Directive
                        Token   No :03  |          <  ------->  |                Operator ,  LSHFT
                        Token   No :04  |   iostream  ------->  |                 Library
                        Token   No :05  |          >  ------->  |                Operator ,  RSHFT
                        Token   No :06  |          #  ------->  |          Special Symbol
                        Token   No :07  |     define  ------->  |  Pre-Processor Directive
                        Token   No :08  |      LIMIT  ------->  |                Variable ,  POINTER TO SYMBOL TABLE
                        Token   No :09  |          5  ------->  |                 Integer
                        Token   No :10  |      using  ------->  |                Keywords
                        Token   No :11  |  namespace  ------->  |                Keywords
                        Token   No :12  |        std  ------->  |                Keywords
                        Token   No :13  |          ;  ------->  |          Special Symbol
                        Token   No :14  |        int  ------->  |                Keywords
                        Token   No :15  |       main  ------->  |                Keywords
                        Token   No :16  |          (  ------->  |          Special Symbol
                        Token   No :17  |          )  ------->  |          Special Symbol
                        Token   No :18  |          {  ------->  |          Special Symbol
                        Token   No :19  |          D  ------->  |                Variable ,  POINTER TO SYMBOL TABLE
                        Token   No :20  |        for  ------->  |                Keywords
                        Token   No :21  |        int  ------->  |                Keywords
                        Token   No :22  |          A  ------->  |                Variable ,  POINTER TO SYMBOL TABLE
                        Token   No :23  |          ,  ------->  |          Special Symbol
                        Token   No :24  |          B  ------->  |                Variable ,  POINTER TO SYMBOL TABLE
                        Token   No :25  |          ;  ------->  |          Special Symbol
```



```
                        Token   No :26  |        cin  ------->  |                Keywords
                        Token   No :27  |         >>  ------->  |                Operator ,  INS
                        Token   No :28  |          A  ------->  |                Variable ,  POINTER TO SYMBOL TABLE
                        Token   No :29  |         >>  ------->  |                Operator ,  INS
                        Token   No :30  |          B  ------->  |                Variable ,  POINTER TO SYMBOL TABLE
                        Token   No :31  |          ;  ------->  |          Special Symbol
                        Token   No :32  |       cout  ------->  |                Keywords
                        Token   No :33  |         <<  ------->  |                Operator ,  EXTR
                        Token   No :34  |          A  ------->  |                Variable ,  POINTER TO SYMBOL TABLE
                        Token   No :35  |          *  ------->  |                Operator ,  MUL
                        Token   No :36  |          B  ------->  |                Variable ,  POINTER TO SYMBOL TABLE
                        Token   No :37  |          ;  ------->  |          Special Symbol
                        Token   No :38  |     return  ------->  |                Keywords
                        Token   No :39  |          0  ------->  |                 Integer
                        Token   No :40  |          ;  ------->  |          Special Symbol
                        Token   No :41  |          }  ------->  |          Special Symbol


        PRESS TYPE `EXIT` TO CLOSE WINDOW.
        NOTE: AN OUTPUT FILE WILL BE GENERATED IN THE SAME FOLDER AS `Output.txt`
EXIT
PS C:\Users\Admin\Desktop\SS_L2>
```