|  |
| --- |
| **Lab 6 : Lexical analyzer** |

Exercise 1

Extend your lexical analyser to ignore comments, defined as follows:

* 1. A comment begins with // and includes all characters until the end of that line.
  2. A comment begins with /\* and includes all characters until ends at \*/ or otherwise end of that file.

**TASK\_01:**

#include<iostream>

#include<fstream>

#include<string>

using namespace std;

int main(){

ifstream input;

input.open("sample.txt");

if(!input){

cout<<"unable to open file"<<endl;

exit(1);

}

string str,str1,temp,temp2;

while(getline(input,str)){

str1+=str;

for(int i=0;i<str.length();i++){

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

for(int j=i; j<str.length();j++)

i++;

}

else

temp+=str[i];

}

}

int i=0;

while(i<temp.length()){

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

for(int j=i; j<temp.length();j++){

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

i++;

break;

}

i++;

}

}

else

temp2+=temp[i];

i++;

}

input.close();

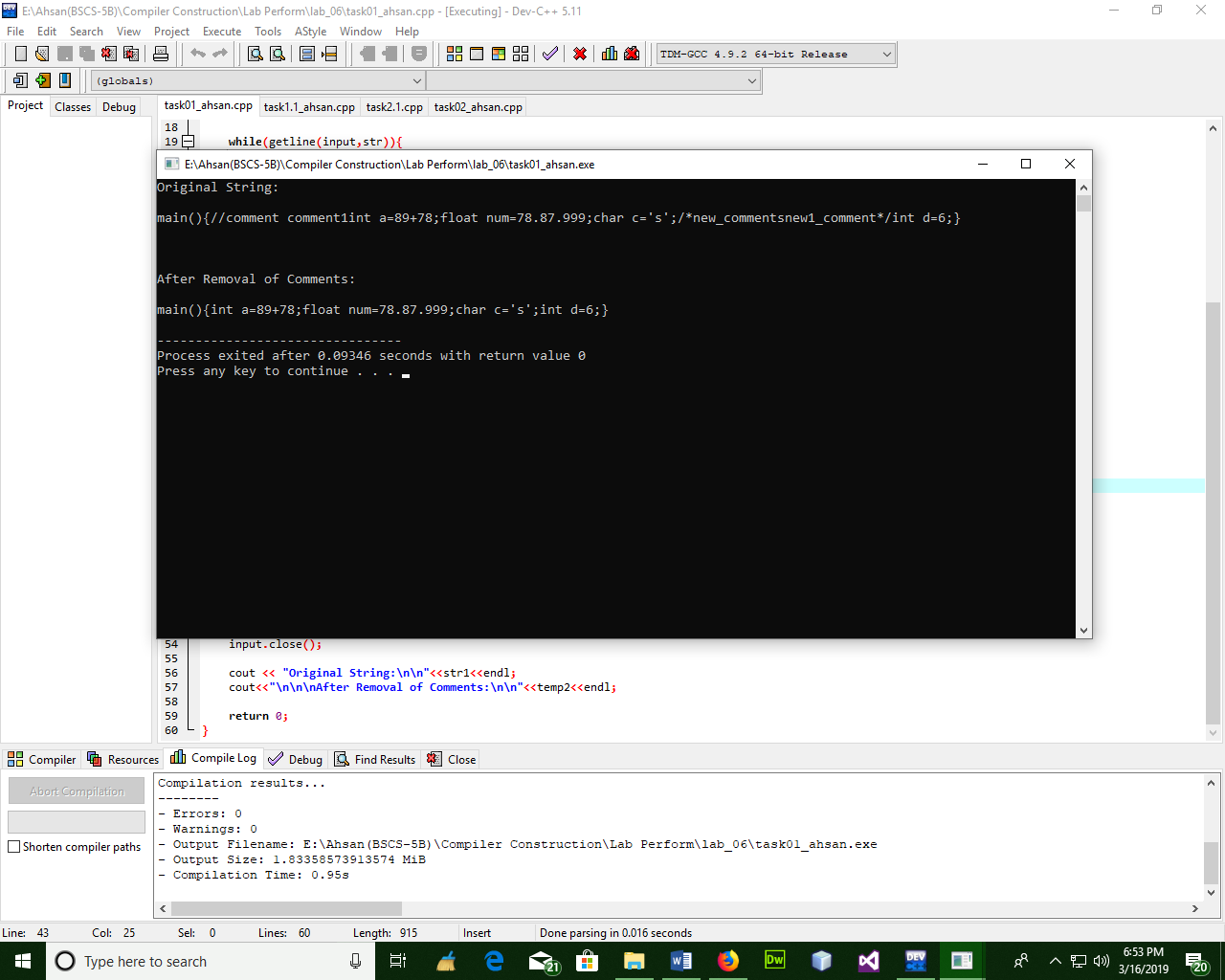
cout << "Original String:\n\n"<<str1<<endl;

cout<<"\n\n\nAfter Removal of Comments:\n\n"<<temp2<<endl;

return 0;

}

**SCREENSHOT:**



Exercise 2

Write a program to generate Token set of given Lexemes and store these token set in a file. Lexical analyzer that can handle following specification.

|  |  |
| --- | --- |
| Identifier | A-Z a-z \_ |
| Keywords | For, if, else,while, switch, case |
| Relational operators | <,>,<=,>=,<> |
| Mathematical operator | +,-,/,\*,^ |
| Logical operator | AND OR NOT |
| Punctuation | () [] {} , . ; : “” |
| Increament decreament | ++ -- |
| Literals | “hello” |
| Data types | int float double bool |

**TASK\_02:**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

int charconst (int cs, char c) {

int TT [4][2] = {{2,1},{2,3},{3,3},{3,3}};

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

return TT[cs][0];

else if (c >= '\\')

return TT [cs][1];

else

return 3;

}

bool charconst (string w) {

int cs=0, fs=2;

int i=0;

for (i=1; i<w.length()-1; i++) {

cs= charconst (cs,w[i]);

}

if (cs==fs && w[0]==39 && w[w.length()-1]==39)

return true;

else

return false;

}

int fp (int cs, char c) {

int fp [4][2] = {{0, 1}, {2, 3}, {2, 3}, {3,3}};

if (c == '0' || c == '1' ||c == '2' ||c == '3' ||c == '4' ||c == '5' ||c == '6' ||c == '7' ||c == '7' ||c == '8' || c == '9')

return fp[cs][0]; //[1][0]

else if (c=='.')

return fp[cs][1]; //[0][1]

else

return 0;

}

bool fp (string w) {

int cs=0, fs=2;

int i=0;

while (w[i]!=NULL) {

cs= fp (cs,w[i]);

i++;

}

if (cs==fs)

return true;

else

return false;

}

int integer (int cs, char c) {

int TT [4][2] = {{1, 2}, {3, 2}, {3, 2}, {3,3}};

if (c == '+' || c== '-')

return TT [cs][0];

else if (c == '0' || c == '1' ||c == '2' ||c == '3' ||c == '4' ||c == '5' ||c == '6' ||c == '7' ||c == '7' ||c == '8' || c == '9')

return TT[cs][1];

else

return 3;

}

bool integer (string w) {

int cs=0, fs=2;

int i=0;

while (w[i]!=NULL) {

cs= integer (cs,w[i]);

i++;

}

if (cs==fs)

return true;

else

return false;

}

int strng (int cs, char x, int i) {

int strng [7][6]={{1,6,6,6,6,6},{6,2,3,6,6,6},{6,2,3,6,5,6},{6,6,6,4,6,6},{6,2,6,6,5,6},{6,6,6,6,6,6},{6,6,6,6,6,6}};

if(x==34 && i==0)

return strng [cs][0];

else if ((x >= 35 && x <= 126||x==32||x==33) && i!=0)

return strng [cs][1];

else if (x == 92)

return strng [cs][2];

else if (x == 116 || x == 110)

return strng [cs][3];

else if (x == 34)

return strng [cs][4];

else

return strng [cs][5];

}

bool strng (string w) {

int cs=0, fs=5;

int i=0;

while (w[i]!=NULL) {

cs= strng (cs,w[i],i);

i++;

}

if (cs==fs)

return true;

else

return false;

}

bool datatypes (string w) {

string dt[] = {"int", "char", "double", "float", "bool", "string"};

for (int i=0; i<6; i++) {

if (w==dt[i])

return true;

}

return false;

}

bool keyword (string w) {

string keywords[] = {"using", "break", "continue", "void", "return", "true", "false", "new", "NULL", "public", "protect", "private", "struct",

"if", "else", "elseif", "do", "while", "for"};

for (int i=0; i<19; i++) {

if (w==keywords[i])

return true;

}

return false;

}

int TTidkw (char c, int s) {

int TT[3][4] = {{1, 2, 1, 2}, {1, 1, 1, 2}, {2, 2, 2, 2}};

if (c == '\_')

return TT[s][0];

else if (c >= '0' && c <= '9')

return TT[s][1];

else if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

return TT[s][2];

else

return TT[s][3];

}

int identifier (string w) {

int cur = 0;

for (int i=0; i<w.length(); i++) {

cur = TTidkw(w[i], cur);

}

return cur;

}

bool logicaloperator (string w) {

string lo[] = {"&&", "||", "!"};

for (int i=0; i<3; i++) {

if (w==lo[i])

return true;

}

return false;

}

int ArithmaticOP (string w) {

string ao[] = {"+", "-", "\*", "/", "%"};

for (int i=0; i<5; i++) {

if (w==ao[i])

return true;

}

return false;

}

int RelationalOP (string w) {

string ro[] = {"<", ">", "<=", ">=", "=", "!="};

for (int i=0; i<6; i++) {

if (w==ro[i])

return true;

}

return false;

}

int Punctuator (string w) {

string punc[] = {"[", "]", "(", ")", "{", "}", ".", ":", ";", "#"};

for (int i=0; i<10; i++) {

if (w==punc[i])

return true;

}

return false;

}

int AssignOP (string w) {

string aso[] = {"+=", "-=", "==", "/=", "%=", "\*="};

for (int i=0; i<6; i++) {

if (w==aso[i])

return true;

}

return false;

}

int incdec (string w) {

string incdec[] = {"++", "--"};

for (int i=0; i<2; i++) {

if (w==incdec[i])

return true;

}

return false;

}

int main() {

string a, word, t, com;

int line=0;

ifstream inf;

inf.open("data.txt");

while(!inf.eof())

{

line++;

getline(inf,a);

for (int i=0; i<=a.length(); i++) {

if (a[i] == '"') {

t += a[i];

i++;

while (a[i] != '"') {

t += a[i];

i++;

}

t += a[i];

i++;

cout<<t;

if (strng(t)==1)

cout<<"\t:\tString Constant at line no. " <<line <<endl;

t = "";

}

if (a[i]==' ' || a[i]=='\n' || a[i]=='\t' || a[i]=='\0') {

if (word != "") {

cout << word << "\t:\t";

if (keyword(word)==1)

cout<<"Keyword at line no. " <<line <<endl;

else if (datatypes(word)==1)

cout<<"Data Type at line no. " <<line <<endl;

else if (identifier(word)==1)

cout<<"Identifier at line no. " <<line <<endl;

else if (incdec(word) == 1)

cout<< "INCDEC at line no. " <<line <<endl;

else if (logicaloperator(word)==1)

cout<<"Logical Operator at line no. " <<line <<endl;

else if (ArithmaticOP(word) == 1)

cout<< "Arithmetic Operator at line no. " <<line <<endl;

else if (RelationalOP(word) == 1)

cout<< "Relational Operator at line no. " <<line <<endl;

else if (RelationalOP(word) == 1)

cout<< "Relational Operator at line no. " <<line <<endl;

else if (charconst(word)==1)

cout<< "Character Constant at line no. " <<line <<endl;

else if (strng(word)==1)

cout<< "String Constant at line no. " <<line <<endl;

else if (integer(word)==1)

cout<< "Integer Constant at line no. " <<line <<endl;

else if (fp(word)==1)

cout<< "Floating Point Constant at line no. " <<line <<endl;

else if (AssignOP(word) == 1)

cout<< "Assignment Operator at line no. " <<line <<endl;

else if (Punctuator(word) == 1)

cout<< "Punctuator at line no. " <<line <<endl;

else

cout<<"Error at line no. at line no. " <<line <<endl;

word= "";

}

}

else

word = word + a[i];

}

}

inf.close();

}

**SCREENSHOT:**

