CD MINI PROJECT

TOKEN GENERATOR

FINAL LEXICAL ANALYSIS

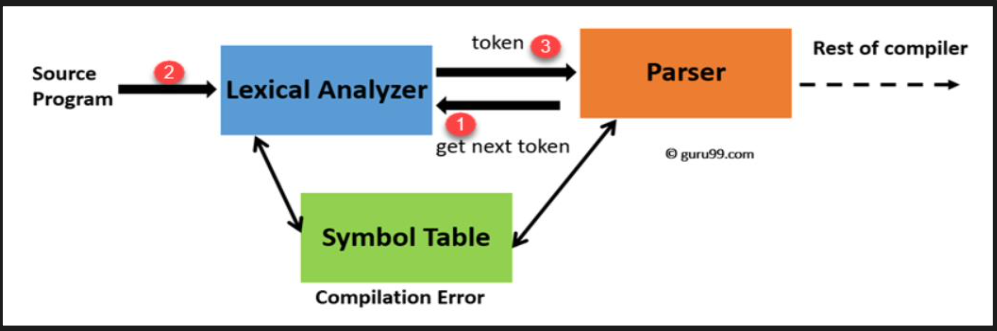
UTKARSH SINHA , MALLIKA SHARDA , ADITYA SINHA

INTRODUCTION

**LEXICAL ANALYSIS** is the very first phase in the compiler designing. It takes the modified source code which is written in the form of sentences. In other words, it helps you to convert a sequence of characters into a sequence of tokens. The lexical analyser breaks this syntax into a series of tokens. It removes any extra space or comment written in the source code.

Programs that perform lexical analysis are called lexical analysers. A lexical analyser contains tokenizer or scanner. If the lexical analyser detects that the token is invalid, it generates an error. It reads character streams from the source code, checks for legal tokens, and pass the data to the syntax analyser when it demands.

ARCHITECTURE AND WORKING



Code:

#include <bits/stdc++.h>

#include <regex>

#include<time.h>

#include <iterator>

#define deb(x) cout<<#x<<" = "<<x<<endl

using namespace std;

map<string,string> Make\_Regex\_Map(){

map<string,string> my\_map {

{ "\\;|\\{|\\}|\\(|\\)|\\,|\\#", "Special Symbol"},

{ "int|char|float|bool|cin|cout|main|using|namespace|std","Keywords"},

{ "\\include|define", "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;

}

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;

}

string tell\_Lexeme(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 return "";

}

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 ){

if(!(match->second.first==" ")&&!(match->second.first=="//")){

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;

}

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;

}

count++;

}

else{

if(match->second.second=="Operator"){

cout.width(40);

string op=tell\_Lexeme(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;

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";

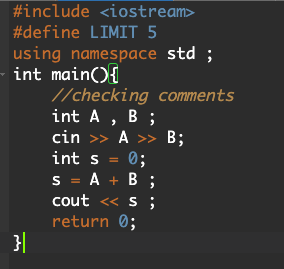
}

return 0;

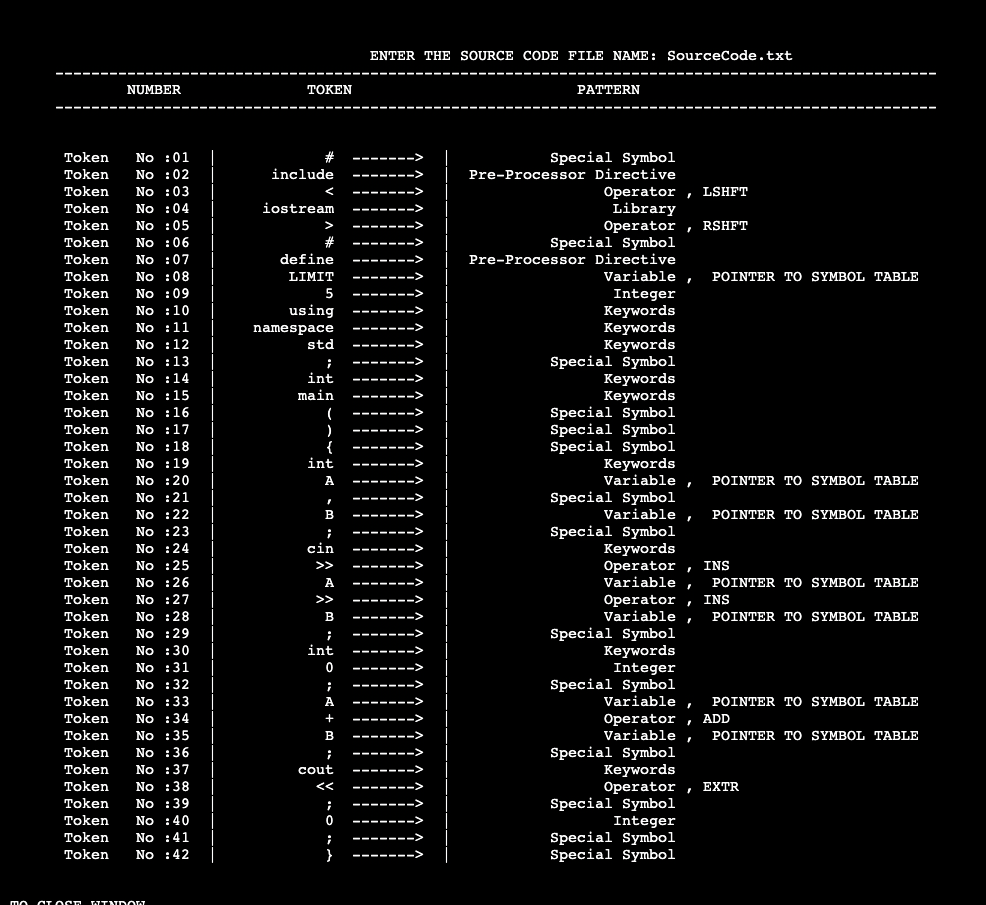
}

Source code (SourceCode.txt)

A program in C++ to take 2 integers as input and print the sum

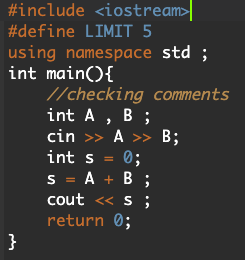


Corresponding output:



Source code 2 (code2.txt)

A c++ program to print a pattern using nested loop



Corresponding output:

