#include<iostream>

#include<regex>

#include<vector>

#include<fstream>

using namespace std;

string stripL(string );

string stripR(string );

string strip(string );

string getSeperatorName(string );

string getType(string );

const string separators = "[=<>\\(\\)\\{\\},\\.;\\[\\]]|(\\s)+";

const string keywords = "(auto)|(break)|(case)|(char)|(const)|(continue)|(default)|(do)|(double)|(else)|(enum)|(extern)|(float)|(for)|(goto)|(if)|(int)|(long)|(register)|(return)|(short)|(signed)|(sizeof)|(static)|(struct)|(switch)|(typedef)|(union)|(unsigned)|(void)|(volatile)|(while)";

const string number = "0|(-?[1-9][0-9]\*)";

const string lattersValidName = "[A-Za-z\_][A-Za-z0-9\_]\*";

int main(){

string fileName="sourceCode.c";

ifstream fs;

fs.open(fileName);

if (!fs) {

cout << "Unable to open file";

exit(1); // terminate with error

}else{ cout<<"File open success."<<endl; }

bool multilineCommentOut{false};

string line;

smatch match;

vector<pair<string,string>> tokens;

while(getline(fs,line)){

line = strip(line);

**///Preprocessor**

if(line[0]=='#'){

tokens.push\_back({"PREPROCESSOR","#"});

line = line.substr(1);

if(regex\_search(line,match,regex("<|(\\s)+"))){

tokens.push\_back({"PREPROCESSOR\_TYPE",match.prefix()});

tokens.push\_back({getSeperatorName(match.str()),match.str()});

line=match.suffix();

}

if(regex\_search(line,match,regex(">|(\\s)+"))){

if(match.str()==">"){

tokens.push\_back({"HEADER\_FILE",match.prefix()});

tokens.push\_back({getSeperatorName(match.str()),match.str()});

}else{

tokens.push\_back({"CONSTANT\_IDENTIFER",match.prefix()});

tokens.push\_back({getSeperatorName(match.str()),match.str()});

tokens.push\_back({"CONSTANT\_VALUE",match.suffix()});

}

}continue;

}

start:

**///Skipping line checking or Comment out part**

if(multilineCommentOut){

if(regex\_search(line,match,regex("\\\*/"))){

line=match.suffix();

multilineCommentOut=false;

goto start;

}continue;

}else if(line==""||regex\_match(line,regex("^//.\*"))){ continue; }

else if(regex\_search(line,match,regex("//"))){

line = strip(match.prefix());

}else if(regex\_search(line,match,regex("/\\\*"))){

line = match.prefix();

multilineCommentOut=true;

string commentString=match.suffix();

if(regex\_search(commentString,match,regex("\\\*/"))){

line+=match.suffix();

multilineCommentOut=false;

goto start;

}if(line==""){ continue; }

}if(line==""){ continue; }

**///Separator finding**

if(regex\_search(line,match,regex(separators))){

if(match.prefix()!=""){

tokens.push\_back(make\_pair(getType(match.prefix()),match.prefix()));

}tokens.push\_back(make\_pair(getSeperatorName(match.str()),match.str()));

line=match.suffix();

goto start;

}

}cout<<endl<<"\*\*\*\*\*\*\*\*\* Tokens are \*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;

for(pair<string,string> token : tokens){

cout<<" <'"<<token.first<<"', '"<<token.second<<"'> "<<endl;

}cout<<endl;

}

string getSeperatorName(string separator){

if(separator=="("){

return "OPEN\_PARANTHESES";

}else if(separator==")"){

return "CLOSE\_PARANTHESES";

}else if(separator=="{"){

return "OPEN\_CURLY\_BRACES";

}else if(separator=="}"){

return "CLOSE\_CURLY\_BRACES";

}else if(separator=="["){

return "OPEN\_SQUARE\_BRAKET";

}else if(separator=="]"){

return "CLOSE\_SQUARE\_BRAKET";

}else if(separator=="<"){

return "OPEN\_ANGULAR\_BRAKET";

}else if(separator==">"){

return "CLOSE\_ANGULAR\_BRAKET";

}else if(separator==","){

return "COMA\_DELIMATOR";

}else if(separator=="."){

return "DOT\_OPERATOR";

}else if(separator==";"){

return "SEMECLONE";

}else if(separator=="="){

return "ASSIGNMENT\_OPERATOR";

}else{

return "SPACES";

}

}

string getType(string str){

if(regex\_match(str,regex(keywords))){

return "KEYWORD";

}else if(regex\_match(str,regex(lattersValidName))){

return "IDENTIFIRE";

}else if(regex\_match(str,regex(number))){

return "CONSTANT\_NUMBER";

}else{

return "UNKNOWN";

}

}

string stripL(string input\_str){

int starting\_pointer=0;

while(input\_str[starting\_pointer]==' '){

starting\_pointer++;

}

return input\_str.substr(starting\_pointer);

}

string stripR(string input\_str){

int ending\_pointer=input\_str.size()-1;

while(input\_str[ending\_pointer]==' '){

ending\_pointer--;

}

return input\_str.substr(0,ending\_pointer+1);

}

string strip(string input\_str){

string part = stripL(input\_str);

part = stripR(part);

return part;

}