# Department of Computing

**CS 354: Compiler Construction**

**Class:** BSCS-6C

# Lab [04]: Lexical Analysis – Part II

**Date: 29th** Oct, 2019

**Time:** [2:00pm – 5:00pm]

# Instructor: Dr. Ayesha Saadia

# Lab Engineer: M. Danyal Sadiq

# Group Lab

Ahmad Amjad Mughal

Abdul Ghaffar Kalhoro

**BSCS-6C**

1. **Create a Symbol Table with the following fields:**
   1. **Token ID and its attribute value i.e. the Lexeme**
   2. **Initialize your symbol table with the C language keywords**

**Code**

void initialize\_symbol\_table()

{

//initializing symbol table with token ID as CAPITAL VALUE OF keyword.

symbol\_table["BREAK"].push\_back("break");

symbol\_table["CASE"].push\_back("case");

symbol\_table["CHAR"].push\_back("char");

symbol\_table["CONST"].push\_back("const");

symbol\_table["CONTINUE"].push\_back("continue");

symbol\_table["DEFAULT"].push\_back("default");

symbol\_table["DOUBLE"].push\_back("double");

symbol\_table["ELSE"].push\_back("else");

symbol\_table["ENUM"].push\_back("enum");

symbol\_table["EXTERN"].push\_back("extern");

symbol\_table["FLOAT"].push\_back("float");

symbol\_table["FOR"].push\_back("for");

symbol\_table["GOTO"].push\_back("goto");

symbol\_table["IF"].push\_back("if");

symbol\_table["INT"].push\_back("int");

symbol\_table["LONG"].push\_back("long");

symbol\_table["RETURN"].push\_back("return");

symbol\_table["SHORT"].push\_back("short");

symbol\_table["STATIC"].push\_back("static");

symbol\_table["STRUCT"].push\_back("struct");

symbol\_table["SWITCH"].push\_back("switch");

symbol\_table["VOID"].push\_back("void");

symbol\_table["WHILE"].push\_back("while");

}

bool checkKeyWord()

{

for(int i=0;i<LENGTH;i++)

{

if(identifier.compare(keywordsTemp[i])==0)

{

return true;

}

}

return false;

}

//Automata implementation of Relational Operators, Punctuations, Arithematic Operators

Token isToken(string line){

num = "";

identifier = "";

Token final;

final.token\_id = "";

char state = '1';

while(line.length() > lIndex){

char ch = line[lIndex];

lIndex = lIndex + 1;

//dfa for this

switch(state){

case '1':

if(ch == '<')

state = '2';

else if(ch == '>')

state = '3';

else if(ch == '=')

state = '5';

else if (ch == '!')

state = '4';

else if (ch == '\*')

state = 'a';

else if (ch == '/')

state = 'b';

else if (ch == '%')

state = 'c';

else if (ch == '+')

state = 'd';

else if (ch == '-')

state = 'f';

else if (ch == '(')

state = 'g';

else if (ch == ')')

state = 'h';

else if (ch == '{')

state = 'i';

else if (ch == '}')

state = 'j';

else if (ch == '[')

state = 'k';

else if (ch == ']')

state = 'l';

else if (ch == ';')

state = 'm';

else if (ch == ':')

state = 'n';

else if (ch == ',')

state = 'o';

else if (ch == '.')

state = 'p';

else if(isdigit(ch)){ //checking for number

state = 'q';

num += ch;

}

else if( (isalpha(ch)) || ch == '\_'){

state = 'w';

identifier+=ch;

}

else {

//lIndex-=1;

return final;

}

break

case '2':

if (ch == '=')

{

final.token\_id = "LE";

final.lexeme = "<=";

countRELOP +=1;

return final;

}

else

{

final.token\_id = "LT";

final.lexeme = "<";

countRELOP +=1;

lIndex-=1;

return final;

}

break;

case '3':

if(ch == '=')

{

final.token\_id = "GE";

final.lexeme = ">=";

countRELOP +=1;

return final;

}

else{

final.token\_id = "GT";

final.lexeme = ">";

countRELOP +=1;

lIndex-=1;

return final;

break;

case '4':

if(ch == '=')

{

final.token\_id = "NE";

final.lexeme = "!=";

countRELOP +=1;

return final;

}

else{

lIndex-=1;

return final;

}

break;

case '5':

if(ch == '=')

{

final.token\_id = "EQ";

final.lexeme = "==";

countRELOP +=1;

return final;

}

else

{

final.token\_id = "OP";

final.lexeme = "=";

countPUNC +=1;

lIndex-=1;

return final;

}

break;

case 'a':

final.token\_id = "MULT";

final.lexeme = "\*";

countAO += 1;

lIndex-=1;

return final;

break;

case 'b':

final.token\_id = "DIV";

final.lexeme = "/";

countAO += 1;

lIndex-=1;

return final;

break;

case 'c':

final.token\_id = "MOD";

final.lexeme = "%";

countAO += 1;

lIndex-=1;

return final;

break;

case 'd':

if (ch == '+')

{

final.token\_id = "INC";

final.lexeme = "++";

countAO +=1;

return final;

}

else

{

final.token\_id = "PL";

final.lexeme = "+";

countAO +=1;

lIndex-=1;

return final;

}

break;

case 'f':

if (ch == '-')

{

final.token\_id = "DEC";

final.lexeme = "--";

countAO +=1;

return final;

}

else

{

final.token\_id = "MINUS";

final.lexeme = "-";

countAO +=1;

lIndex-=1;

return final;

}

break;

case 'g':

final.token\_id = "LPAREN";

final.lexeme = "(";

countPUNC += 1;

lIndex-=1;

return final;

break;

case 'h':

final.token\_id = "RPAREN";

final.lexeme = ")";

countPUNC += 1;

lIndex-=1;

return final;

break;

case 'i':

final.token\_id = "MLPAREN";

final.lexeme = "{";

countPUNC += 1;

lIndex-=1;

return final;

break;

case 'j':

final.token\_id = "MRPAREN";

final.lexeme = "}";

countPUNC += 1;

lIndex-=1;

return final;

break;

case 'k':

final.token\_id = "SQLPAREN";

final.lexeme = "[";

countPUNC += 1;

lIndex-=1;

return final;

break;

case 'l':

final.token\_id = "SQRPAREN";

final.lexeme = "]";

countPUNC += 1;

lIndex-=1;

return final;

break;

case 'm':

final.token\_id = "SCOLON";

final.lexeme = ";";

countPUNC += 1;

lIndex-=1;

return final;

break;

case 'n':

final.token\_id = "COLON";

final.lexeme = ":";

countPUNC += 1;

lIndex-=1;

return final;

break;

case 'o':

final.token\_id = "COMMA";

final.lexeme = ",";

countPUNC += 1;

lIndex-=1;

return final;

break;

case 'p':

final.token\_id = "DOT";

final.lexeme = ".";

countPUNC += 1;

lIndex-=1;

return final;

break;

//for number checking

case 'q':

if (isdigit(ch)){

state = 'q';

num += ch;

}

else if (ch == '.'){

state = 'r';

num += ch;

}

else if (ch == 'E'){

state = 't';

num += ch;

}

else {

final.token\_id = "NUM";

final.lexeme = num;

countNUM += 1;

lIndex-=1;

return final;

}

break;

case 'r':

if (isdigit(ch)){

state = 's';

num += ch;

}

else{

lIndex-=1;

return final;

}

break;

case 's':

if (isdigit(ch)){

state = 's';

num += ch;

}

else if (ch == 'E'){

state = 't';

num += ch;

}

else {

final.token\_id = "NUM";

final.lexeme = num;

countNUM += 1;

lIndex-=1;

return final;

}

break;

case 't':

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

state = 'u';

else if(isdigit(ch))

state = 'v';

else{

lIndex-=1;

return final;

}

break;

case 'u':

if(isdigit(ch))

state = 'v';

else{

lIndex-=1;

return final;

}

break;

case 'v':

if(isdigit(ch))

state = 'v';

else

{

final.token\_id = "NUM";

final.lexeme = num;

countNUM += 1;

lIndex-=1;

return final;

}

break;

case 'w':

if(isdigit(ch) || isalpha(ch) || ch == '\_'){

identifier +=ch;

state = 'w';

}

else

{

if(!checkKeyWord()){

final.token\_id = "ID";

final.lexeme = identifier;

countID += 1;

lIndex-=1;

return final;

}

else{

final.token\_id = "KeyWord";

final.lexeme = identifier;

countKW+=1;

lIndex-=1;

return final;

}

}

break;

}

}

return final;

}

**Output**

****

1. **Implement the *getNextToken()* function for the scanner, and write a driver function to call it and print all tokens in the source program. Your implementation should be able to differentiate between Keywords and Identifiers.**

**Code**

vector<Token> getNextToken(string line){

vector<Token> tokens;

line = line + " ";

while (lIndex < line.length()){

Token temp = isToken(line);

if((temp.token\_id).compare("") != 0){

tokens.push\_back(temp);

}

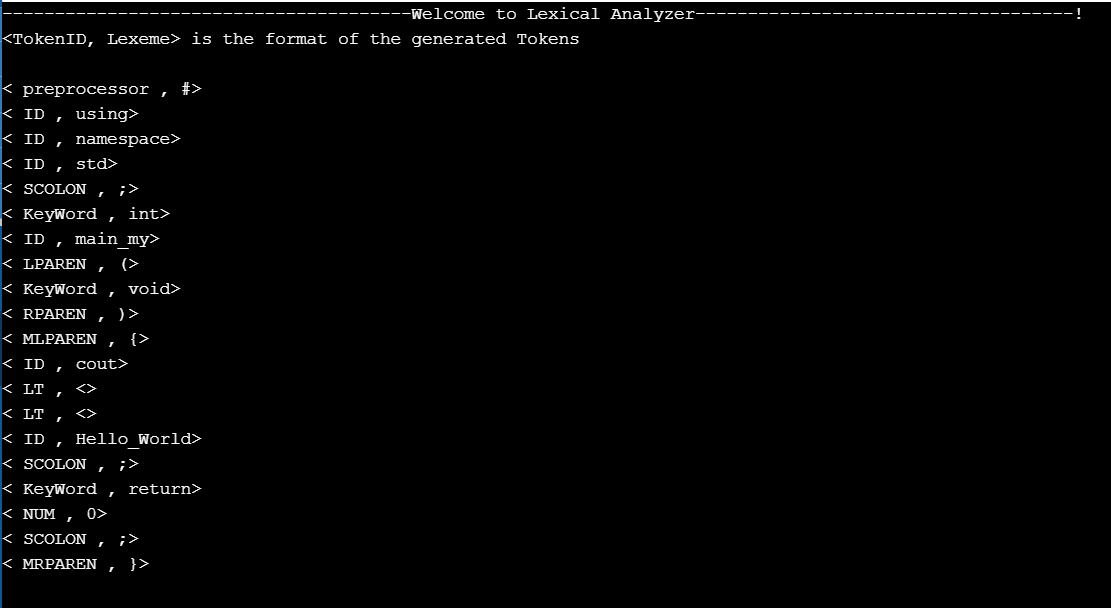
}

return tokens;

}

1. Test your implementation for the following lines of code:
2. int main(void)
3. {
4. printf("Hello, world!**\n**");
5. **return** 0;
6. }

**Result**

****

Total code is provided along with Word file in a shape of source file along with helping files. The required code is given in this document.

**Conclusion**

In this lab we successfully generated tokens of any type of written source code program. In Tabular form we extracted them one by one and categorized them in their specific type and displayed them on command prompt. The functions **get\_Token()** and **symbol\_table()** are implemented and are provided. After doing this lab, we are now able to understand what analyzer understands.