Ahsanullah University of Science & Technology

Department of Computer Science & Engineering

Course No	: CSE4130
Course Title	: Formal Languages and Compilers Lab
Assignment No	: 04

Date of Performance	: 17-01-23
Date of Submission	: 31-01-23

Submitted To	: Mr. Aminur Rahman & Mr. Al Hasib Mahamud

Submitted By-

Group	: A_2
Name	: Md. Nabil Rahman Khan
Id	: 190104044
Section	: A

Code:

```
#include <bits/stdc++.h>
#include <iostream>
#include <fstream>
#include <regex>
#include <string.h>
using namespace std;

bool isKeyword(string str)
                                //check if the given substring is a keyword
or not
{
    string keyword[]=
{"if","else","printf","while","for","do","break,","continue","int","dou
ble","float","return","char","case","long","short","typedef","switch
","unsigned","void","static","struct","sizeof","long","volatile","enu
m","const","bool","union","extern"};
    for(int k=0; k< (sizeof keyword / sizeof keyword[0]); k++)
    {

        if(keyword[k].compare(str)==0)return true;
    }
    return false;
}
bool areBracketsBalanced(string expr)
{
    // Declare a stack to hold the previous brackets.
    stack<char> temp;
    for (int i = 0; i < expr.length(); i++)
    {
        if (temp.empty())
        {

            // If the stack is empty
            // just push the current bracket
            temp.push(expr[i]);
        }
        else if ((temp.top() == '(' && expr[i] == ')')
                 || (temp.top() == '{' && expr[i] == '}')
                 || (temp.top() == '[' && expr[i] == ']'))
        {

            // If we found any complete pair of bracket
            // then pop
            temp.pop();
        }
        else
        {
            temp.push(expr[i]);
        }
    }
    if (temp.empty())
    {

        // If stack is empty return true
        return true;
    }
    return false;
}
bool isPair(string str)
                                //check if the given substring is a keyword
or not
{

    string Pair[]= {"(",")","{","}","[","]"};
    for(int k=0; k< (sizeof Pair / sizeof Pair[0]); k++)
 {

        if(Pair[k].compare(str)==0)return true;
    }
    return false;

}
void errorFinder(regex rgx,string text)
{

    string totalparanthesis;
    string lineno="-1";
    bool parenthesis=true;
    bool semicolon=true;
    string lastwordtochecksemicolon;
    string lastkeyword;
    bool keywordtokrn=true;
    int ifcount=0;
    bool existif=false;
    for( sregex_iterator it(text.begin(), text.end(), rgx), it_end; it
!= it_end; ++it )
    {
        string temp=(*it)[0].str();
        if (temp.find("xac44rk") != string::npos)lineno = temp[0];
        if(isPair(temp)&&parenthesis)
        {
            totalparanthesis=totalparanthesis+" "+temp;
            if(temp.compare("}")==0 )
            {
                if(!areBracketsBalanced(totalparanthesis))
                {
                    cout<<" Misplaced at Line  "<<lineno<<",";
                        parenthesis=false;
                }

            }

        }
        if(temp.compare(lastwordtochecksemicolon)==0 &&
temp.compare(";")==0 && semicolon)
        {
            cout<<" Duplicate token at line   "<<lineno<<", ";
            semicolon=false;
        }


        if(temp.compare(lastkeyword)==0  && keywordtokrn &&
isKeyword(temp))
        {
            cout<<" Duplicate keyword at line   "<<lineno<<", ";
            keywordtokrn=false;
        }



        if(temp.compare("if")==0)
        {
            existif=true;
        }
        if(temp.compare("else")!=0 && existif)existif=false;
        else if(temp.compare("else")==0 && existif==false )
        {
            cout<<" Error at line "<<lineno<<",";
        }
        else if(temp.compare("else")==0 && existif )existif=true;
```

```
            lastkeyword=temp;
        lastwordtochecksemicolon=temp;
    }

}
int main()
{
    fstream file;
    string  text,temp;
    file.open("input.txt", ios::in | ios::app);
    int i=0;
    if (!file.is_open())cout << "No FIle ! Error";
    else
    {
        while (!file.eof())
        {
            i++;
            getline(file, temp);
            text = text + to_string(i)+"xac44rk"+" "+temp + "\n";
        }

        //cout<<text<<endl;

        regex r2("(\\S+)");
        errorFinder(r2,text);

        file.close();


    }
}
```

<u>Input.txt</u>

```
float  x1 = 3.125 ; ; ;

double  f1 ( float  a , int int  x )
{ if (  x <  x1 )
double  z ; ;
else  z = 0.01 ; } }
else return  z ;
}

int  main ( void )
{ { { {
int  n1 ; double  z ;
n1 = 25 ; id z =  f1 (  n1 ) ; }
```

<u>Output.txt</u>

```
 Duplicate token at line   3,  Duplicate keyword at line   5,
Error at line 8, Misplaced at Line  8, Error at line 9,
```

<u>Question:</u>

Suppose, a given C source program has been scanned, filtered, lexically analyzed and tokenized as that were done in earlier sessions. In addition, line numbers have been assigned to the source code lines for generating proper error messages. As the first step to Syntax Analysis, we now perform detection of simple syntax errors like duplication of tokens except parentheses or braces, unbalanced braces or parentheses problem, unmatched ‘else’ problem, etc. Duplicate identifier declarations must also be detected with the help of the Symbol Table