Ahsanullah University of Science and Technology (AUST)
Department of Computer Science and Engineering

Assignment 4

Course No.: CSE4130

Course Title: Formal Languages & Compilers Lab

Date of Submission-12.07.2023

Submitted To- Mr. Md. Aminur Rahman & Iffatur Nessa.

Submitted By-

MD Shihabul Islam Shovo
190204075
Group: B1
Year- 4th
Semester- 1st
Session: Fall'22
Department- CSE

Answer:

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include<iomanip>
#include<algorithm>
using namespace std;
// variable declaration
ifstream rf; ofstream wf;
vector<string> kws = { "auto", "break", "case", "char", "const", "continue", "default",
        "do", "double", "else", "enum", "extern", "float", "for", "goto",
        "if", "inline", "int", "long", "register", "restrict", "return",
        "short", "signed", "sizeof", "static", "struct", "switch", "typedef",
        "union", "unsigned", "void", "volatile", "while", "_Bool", "_Complex",
"_Imaginary" };
vector<string> ids; //stores all the indentifier
string ops = "+-*/%=<>!|&";
string pars = "(){}[]";
string seps = ",;'\"";
string op;
char c;
string s;
struct TokenStruct { // structure of a token
   int no;
   string type;
   string value;
};
vector <TokenStruct> token; // vector of TokenStruct structure
```

```
// User Defined function as needed
int read_file(string filename) {
   // This function will take a file name input from the user and open it in read mode
   rf.open(filename);
   if (!rf) {
      cout << "Error opening file.\n";
      return 1;
   }
   return 0;
}
void plainC() {
   // This plainC() function removes all newlines, extra spaces, and comments from a C source code file
   FILE *readFile,*writeFile;
   char c1='\0', c2 = ' ';
   int line_no = 1;
   readFile = fopen("input.c","r");
   writeFile = fopen("plainC.txt", "w");

   //If file is not created then show this message
   if(!readFile)cout<<"\nFile not found";

   /*if File is created then
    remove the spaces, empty line & comments*/

   else
   {
      c1 = fgetc(readFile); c1 = fgetc(readFile); c1 = fgetc(readFile);
```

```
fprintf(writeFile, "%d ", line_no++);
while((c1 = fgetc(readFile))!= EOF)
{
   if(c1==' '){
      fputc(' ', writeFile);
      while((c1=fgetc(readFile)) == ' ');
   }
   if((c1=='\n')){
      fputc('\n', writeFile);
      fprintf(writeFile, "%d ", line_no++);
      continue;
   }
   if((c1=='/') && ((c2 = fgetc(readFile))== '/')){
      while((c1=fgetc(readFile))!='\n');
      fputc('\n', writeFile);
      fprintf(writeFile, "%d ", line_no++);
   }
   else if((c1=='/') && (c2=='*')){
      c2 = c1; c1 = fgetc(readFile);
      if(c1=='\n'){
         fputc('\n', writeFile);
         fprintf(writeFile, "%d ", line_no++);
      }
      while((c1!='/') && (c2 != '*')) {
         c2 = c1;
         c1 = fgetc(readFile);
         if(c1=='\n'){
            fputc('\n', writeFile);
```

```
                    fprintf(writeFile, "%d ", line_no++);
                } }}
            else{ fputc(c1, writeFile);}
            c2 = c1; }}
    fclose(readFile);
    fclose(writeFile);
}
int isoperator() {
    // isoperator() function will check for an operator
    if (ops.find(c) != string::npos) {
        op += c;
        if ((c = rf.get()) != EOF) {
            isoperator();
        }
        return 1;
    }

    if (!op.empty()) {
        rf.unget();
        return 0;
    }
    return 0;
}
int isprnorsep(string str) {
    // isprnorsep() function will check for a parenthesis or separator
    if (str.find(c) != string::npos) {
        return 1;
    }
```

```
    return 0;
}
int iskeyword() {
    for (const string& keyword : kws) {
        if (s == keyword) {
            return 1;
        }
    }
    return 0;
}
int isidentifier() {
    // isidentifier() function finds the valid keywords also labeled as id and if not valid then labeled as unkn
    for (int i = 1; i < ids.size(); i++) {
        if (s == ids[i]) {
            return 1;
        }
    }

    int len = s.length();
    if (s[0] == '_' || isalpha(s[0])) {
        for (int i = 1; i < len; i++) {
            if (s[i] == '_' || isalnum(s[i])) {
                continue;
            }
            else {
                return 0;
            }
        }
```

```
        ids.push_back(s);
        return 1;
    }
    return 0;
}
int isnumber() {
    // Check if the word is a number or not
    int len = s.length();
    int i, nflag = 0;
    for (i = 0; i < len; i++) {
        if (isdigit(s[i])) {
            nflag = 1;
        }
        else if (s[i] == '.') {
            nflag = 2;
            i++;
            break;
        }
        else {
            return 0;
        }
    }
    if (nflag == 2) {
        while (i < len) {
            if (isdigit(s[i])) {
                nflag = 1;
            }
            else {return 0; }
```

```
            i++;
        }}
    if (nflag == 1) {
        return 1;
    }
    return 0;
}
void insertToken(string type, string str){
    // insert tokens to a vector of structure
    TokenStruct newtoken;
    newtoken.no = token.size() + 1;
    newtoken.type = type;
    newtoken.value = str;

    token.push_back(newtoken);
}
void lexemes() {
    cout<<"Step 1: Intermediate Output: Recognized tokens in the lines of code."<<endl;
    // This function analyzes all the words and finds the lexemes
    // Read a c file to get the source code
    if (read_file("plainC.txt") != 0) {
        cout<<"Error opening file"<<endl;
    }
    wf.open("lexemes.txt");
    c = rf.get(); // to read the first line no value
    s=c; insertToken("lno", s); wf<<"[lno "<<s<<"] ";  s.clear(); cout<<c;
    while ((c = rf.get()) != EOF) {
        if(c=='\n'){
```

```
        cout<<c;
        c = rf.get(); cout<<c;
        while(isdigit(c)) { s+=c; c = rf.get(); cout<<c;}
        insertToken("lno", s);
        wf<<"[lno "<<s<<"] ";
        s.clear();
    }
    if(isspace(c)){
        cout<<c;
        continue;
    }
    // Read letters and store the word
    for (int i = 0; !isspace(c) && !isoperator() && !isprnorsep(pars) && !isprnorsep(seps); i++)
{
        // Store the letters until there is a space, operator, parenthesis, or separator
        // If isoperator() function is called, this will store the operator or consecutive operators
        // Other functi ons will only return a positive value or 1
        s += c;
        c = rf.get();
    }
    if (!s.empty()) {
        if (iskeyword()) {cout<<"kw "<<s<<" "; insertToken("kw", s); wf<<"[kw "<<s<<"] ";} //
insertToken(string , string) receives two string value and insert as token
        else if (isidentifier()) {cout<<"id "<<s<<" "; insertToken("id", s); wf<<"[id "<<s<<"] ";}
        else if (isnumber()) {cout<<"num "<<s<<" "; insertToken("num", s); wf<<"[num
"<<s<<"] ";}
    }
    if (!op.empty()) {
        // If there is an operator stored from the previous call of isoperator() function tokenize
the operator
```

```
            insertToken("op", op); wf<<"[op "<<op<<"] ";
            cout<<"op "<<op<<" "; op.clear(); // clear the operator so that next time it don't contain any value if isoperator() function don't assign any value to op
        }
        else if (isprnorsep(pars)) {
            // Call the isprnorsep() function and tokenize the parenthesis
            s=c; // converts char to string
            insertToken("par", s); wf<<"[par "<<s<<"] ";
            cout<<"par "<<s<<" ";
        }
        else if (isprnorsep(seps)) {
            // Call the isprnorsep() function and tokenize the separator
            s=c;
            insertToken("sep", s); wf<<"[sep "<<s<<"] ";
            cout<<"sep "<<s<<" ";
        }
        s.clear();
    }
    rf.close();
    wf.close();
    cout<<endl;
}
void detectErrors(){
    cout<<"\nStep 2: Detected errors:"<<endl;
    int errors=1;
    string lno="0";
    int sb=0, sc=0, cm=0, kw=0, ifs=0, elf=0, other=0;
    for(int i=0; i<token.size(); i++){
        TokenStruct& t = token[i];
```

```
if(t.type=="lno"){
        if(sb>1)
           cout<<"Error "<<errors++ <<" : Misplaced '{' at line "<<lno<<endl;
        if(sb<-1)
           cout<<"Error "<<errors++ <<" : Misplaced '}' at line "<<lno<<endl;
        lno = t.value;
        sb=sc=cm=kw=other=0;
}
else {
    other++;
    if(other!=0 && t.value!=";") sc=0;
    if(other!=0 && t.value!=",") cm=0;
    if(other!=0 && t.type!="kw") kw=0;
}
if(t.type=="par"){
    if(t.value=="{") sb++;
    else if(t.value=="}") sb--;
}
if(t.type == "sep"){
    if(t.value==";") {
        sc++;
        if(sc>1)
           cout<<"Error "<<errors++ <<" : Duplicate \";\" at line "<<lno<<endl;
    }
    else if(t.value==",") {
        cm++;
        if(cm>1)
           cout<<"Error "<<errors++ <<" : Duplicate \",\" at line "<<lno<<endl;
```

```
        } }
    else if(t.type == "kw" && t.value!="if" && t.value!="else" &&t.value!="for" && t.value!="while"&& t.value!="return"){
        kw++;
        if(kw>1)
            cout<<"Error "<<errors++ <<" : Duplicate keywords at line "<<lno<<endl;
    }
    else if(t.type=="kw" && t.value=="if"){
            ifs=1;
    }
    else if(t.type=="kw" && t.value=="else" && token[i+1].value=="if"){
        if(ifs==0)
            cout<<"Error "<<errors++ <<" : Unmatched 'else if' at line "<<lno<<endl;
        i++; ifs=0; elf=1;
    }
    else if(t.type=="kw" && t.value=="else"){
        if(ifs==0 && elf==0)
            cout<<"Error "<<errors++ <<" : Unmatched 'else' at line "<<lno<<endl;
        ifs=elf=0;
    }}}
// Main function
int main() {
  /*create a file named as "input.txt"
  and put c code in that file to get valid output*/
  plainC(); // removes comments and adds line no
  lexemes(); //step 1 is in function lexemes
  detectErrors(); // step 2
  return 0;
}
```