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

# Assignment 3

Course No.: CSE4130

Course Title: Formal Languages & Compilers Lab

**Date of Submission-14.06.2023**

**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, 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;
```

```
};
struct SymbolTable{ // structure of the symbol table
   int si_no;
   string name, id_type, data_type, scope, value;
};
vector <TokenStruct> token; // vector of TokenStruct structure
vector <SymbolTable> table; // vector of SymbolTable 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
   string filename = "input.c";
   read_file(filename);

   ofstream p2;
   char c, c2 = ' ';
   p2.open("plainC.txt");

   while ((c = rf.get()) != EOF) {
```

```
        if (c == ' ' || c == '\n') {
            p2 << ' ';
            while ((c = rf.get()) == ' ' || c == '\n');
        }
        if ((c == '/') && ((c2 = rf.get()) == '/')) {
            while (((c = rf.get()) != '\n'));
        }
        else if ((c == '/') && (c2 == '*')) {
            while ((c != '/') && (c2 != '*')) {
                c2 = c;
                c = rf.get(); // read a character from a file
            }
        }
        else {
            p2 << c; // store on the file
            //cout << c;
        }
        c2 = c;
    }
    p2.close();
    rf.close();
    //cout<<"\n"<<endl;
}
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);
}
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;
}
int lexemes() {
    // 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) {
        return 1;
    }
    int err=0;
    while ((c = rf.get()) != EOF) {
        // 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 functions will only return a positive value or 1
            s += c;
            c = rf.get();
        }
        if (!s.empty()) {
            if (iskeyword()) insertToken("kw", s); // insertToken(string , string) receives two string
value and insert as token
            else if (isidentifier()) insertToken("id", s);
            else if (isnumber()) insertToken("num", s);
```

```
        }
        if (!op.empty()) {
            // If there is an operator stored from the previous call of isoperator() function tokenize the operator
            insertToken("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);
        }
        else if (isprnorsep(seps)) {
            // Call the isprnorsep() function and tokenize the separator
            s=c;
            insertToken("sep", s);
        }
        s.clear();
    }
    rf.close();
    return 0;
}
bool isdatatype(string str) {
    vector<string> dt = { "int", "float", "double", "char", "bool", "vector", "string" };
    for (const string& datatype : dt) {
        if (str == datatype) {
            return true;
        }
```

```
    }
    return false;
}
void setAttribute(int i, string recentScope){
    //find id the variable already exist in the symbol table and if it has assigned a value
    //then update the value in symbol table
    TokenStruct& t = token[i];
    for(auto& src: table){
        if(t.value==src.name && src.id_type=="var" && recentScope==src.scope){ // recentScope==src.scope is used to check existed variable from the same scope
            t=token[++i];
            if(t.value=="="){
                t=token[++i];
                if(t.type=="num"){
                    src.value = t.value;
                } else t = token[--i];
            }
        }
    }
}
void Insert(vector <TokenStruct> newtoken){ // instance of a vector of structure so that the main variable's values doesn't get manipulated
    // Insert new entry in the symbol table for lexemes
    SymbolTable tb;
    string recentScope, lastScope= "Global"; // initially scope is Global
    string recentDatatype;
    int braces=0;
    for(int i=0; i<token.size(); i++){
        TokenStruct& t = newtoken[i]; // pointer t to indicate a vector of structure's (newtoken) index
```

```
    if(braces==0){ // braces value 0 means it is in Global section
       recentScope = "Global";
    }
    if(t.value == "{") {
       braces++;
       recentScope = lastScope;
    }
    else if(t.value == "}") braces--;
    if(t.type =="kw" && isdatatype(t.value)){ // isdatatype() function to check if it's a data type or not
       tb.si_no = table.size() + 1; // sirial no
       recentDatatype = t.value; // this is used for next variable in a single type
       t = newtoken[++i]; // get the next token from the newtoken vector
       if(t.type == "id"){
          tb.name = t.value; // name
          tb.data_type = recentDatatype; // data type
          tb.scope = recentScope; // scope
          t = newtoken[++i];
          if(t.value == "("){
             tb.id_type = "func"; // id type = func
             recentScope = tb.name;
             lastScope = recentScope; // save the current scope for further use for variables in this scope
             tb.value = "\0"; // no value has for funciton
          }
          else if(t.value == "=" || t.value == ";" || t.value == ")"){ // could be x1 = 121 or x1; or f1(int x1)
             tb.id_type = "var"; // id type = var
             tb.scope = lastScope; // scope
             if(t.value == "="){ // = means a value is assigned for this variable
```

```
                t = newtoken[++i];
                if(t.type == "num"){ // condition to check if assigned value is a num attribute
                   tb.value = t.value; // value of the variable
                }
                else t = newtoken[--i]; // if the if statement is not true then go to the previous
token
              }
            }
            else t = newtoken[--i];

         } table.push_back(tb); // push new values in the vector
      }
      else if(t.type=="id"){
         setAttribute(i, recentScope); // update values of variables
      }
   }
}
bool searchByString(const SymbolTable& obj, const string& value) {
   return obj.name == value;
}
void free(){
   // Delete all the entry from symbol table
   if(table.size()>0){
      table.erase(table.begin(),table.end());
      cout<<"--All entry cleared successfully."<<endl<<endl;
   }
   else cout<<"--Symbol table is already empty."<<endl<<endl;
}
void lookUp(){
```

```
   // lookUp() function search for a name in the symbol table
   if(table.size()>0){
      string searchName;
      cout<<"Enter a name to search: ";
      cin>>searchName;
      auto stringResult = find_if(table.begin(), table.end(),
         [searchName](const SymbolTable& obj) { return searchByString(obj, searchName); });
      if (stringResult != table.end()) {
         cout << "--Result: The searched name's SI.No is: " << stringResult->si_no << endl<<endl;
      }
      else {
         cout << "--Error: Name \""<<searchName <<"\" doesn't exist on the symbol table!" << endl<<endl;
      }
   }
   else cout<<"--Symbol table is empty."<<endl<<endl;
}
void displayTable(){
   if(table.size()>0){
      cout <<left<< setw(20) << "SI.No" << setw(20) << "Name" << setw(20) << "ID Type" << setw(20) << "Data Type" << setw(20) << "Scope" << setw(20) << "Value" << endl;
      cout<<"--------------------------------------------------------------------------------------------------------------------"<<endl;
      for(const auto& t: table){
         cout <<left<< setw(20) << t.si_no <<setw(20)<<  t.name <<setw(20)<<  t.id_type <<setw(20)<<  t.data_type <<setw(20)<<  t.scope <<setw(20)<< t.value << endl;
      }
      cout<<endl;
   }
```

```
   else cout<<"--Symbol table is empty."<<endl<<endl;
}
void displayLexemes(){
   cout << left << setw(7) <<"No" << setw(12)<< "Type" <<setw(12)<< "Value" << "|| "
        << setw(7) <<"No" << setw(12)<< "Type" <<setw(12)<< "Value" << "|| "
        << setw(7) <<"No" << setw(12)<< "Type" <<setw(12)<< "Value" << endl;
        cout<<"-------------------------------------------------------------------------------------------
"<<endl;
   for(int i=0; i<token.size(); i++){
      TokenStruct& t = token[i];
      cout <<left<<setw(7)<< t.no <<setw(12)<<  t.type <<setw(12)<< t.value << "|| ";
      t = token[++i];
      cout << setw(7)<< t.no <<setw(12)<<  t.type <<setw(12)<< t.value << "|| ";
      t = token[++i];
      cout << setw(7)<< t.no <<setw(12)<<  t.type <<setw(12)<< t.value <<endl;
   }
   cout<<endl;
}
void userChoice(){
   int choice;
   while(1){
      cout<<"Choose an option: " << endl
         << " 1. Lookup: Search for a name on the symbol table. " << endl
         << " 2. Free: remove all entries." << endl
         << " 3. Display Symbol table" << endl
         << " 4. Display the lexemes" << endl
         << " 5. Exit" << endl
         << "\nEnter your choice: ";
         cin>>choice;
```

```
        if(choice == 1) lookUp();
        else if(choice == 2) free();
        else if(choice == 3) displayTable();
        else if(choice == 4) displayLexemes();
        else exit(0);
    }
}
// Main function
int main() {
    plainC();
    if (lexemes() != 0) {
        return 1;
    }
    Insert(token); // // Insert new entry in the symbol table
    userChoice();

    return 0;
}
```