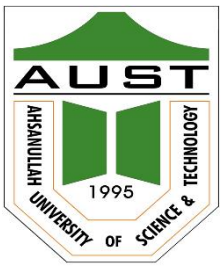

Ahsanullah University of Science and Technology (AUST)

Department of Computer Science and Engineering

Assignment 3

Course No.: CSE4130

Course Title: Formal Languages & Compilers Lab

Date of Submission-
12/06/2023

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

Submitted By-
MD Fardin Jaman Aranyak
190204093
Group: B2
Year- 4th
Semester- 1st
Session: Fall'22
Department- CSE

```
//Fardin Jaman Aranyak
//ID:190204093
//Imagination Better Than Knowledge

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <algorithm>
#include <iomanip>
#include<bits/stdc++.h>

// Structure to represent symbol table entry
struct SymbolEntry {
   int slNo;
   std::string name;
   std::string idType;
   std::string dataType;
   std::string scope;
   std::string value;
};

// Global variables
std::vector<SymbolEntry> symbolTable;

// Function declarations
void display();
void lookup();
void freeSymbolTable();
```

```
void setAttribute();
void insert();
using namespace std;
int main() {
   std::ifstream file("input.txt");
   std::string lexemes;
   if (file.is_open()) {
      std::string line;
      while (std::getline(file, line)) {
         lexemes += line;
      }
      file.close();
   } else {
      std::cout << "Failed to open the input file." << std::endl;
      return 1;
   }

   std::string copyLexemes;
   for (size_t i = 0; i < lexemes.length(); i++) {
      if (lexemes[i] == '[') {
         copyLexemes += lexemes[i];
         copyLexemes += ' ';
      } else if (lexemes[i] == ']') {
         copyLexemes += ' ';
         copyLexemes += lexemes[i];
      } else {
         copyLexemes += lexemes[i];
      }
   }
```

```
    std::vector<std::string> lexemesList;
    std::string token;
    for (size_t i = 0; i < copyLexemes.length(); i++) {
        if (copyLexemes[i] == ' ') {
            if (!token.empty()) {
                lexemesList.push_back(token);
                token.clear();
            }
        } else {
            token += copyLexemes[i];
        }
    }
    cout<<"Input Stream:"<<endl;
    cout<<copyLexemes<<endl;
    std::vector<std::string> tokenToBeRemove = {"kw", "op", "num", "sep", "par", "brc"};
    std::vector<std::string> dataType = {"double", "int", "float"};
    std::vector<std::vector<std::string>> idNamesWithDataType;
    std::vector<std::vector<std::string>> idNamesWithType;
    std::vector<std::vector<std::string>> idNamesWithValue;

    for (auto it = lexemesList.begin(); it != lexemesList.end(); ) {
    if (std::find(tokenToBeRemove.begin(), tokenToBeRemove.end(), *it) != tokenToBeRemove.end()) {
        it = lexemesList.erase(it);
    } else {
        ++it;
    }
}
    cout<<"Step 1: "<<endl;
```

```
   for(int i =0;i<=lexemesList.size();i++){
      cout<<lexemesList[i]<<" ";
   }
   int scopeFlag = 0;
   std::string scope;
   for (size_t i = 0; i < lexemesList.size(); i++) {
      if (lexemesList[i] == "id" && lexemesList[i + 4] == "(") {
         scope = lexemesList[i + 1];
         scopeFlag = 1;
      } else if (lexemesList[i] == "id" && lexemesList[i + 1] == "main" && lexemesList[i + 4] == "(") {
         scope = "main";
         scopeFlag = 1;
      } else if (lexemesList[i] == "}") {
         scopeFlag = 0;
      } else if (scopeFlag == 0) {
         scope = "global";
      }
      if (lexemesList[i] == "id" && std::find(dataType.begin(), dataType.end(), lexemesList[i - 3]) !=
dataType.end()) {
         if (lexemesList[i + 1] == "main") {
            idNamesWithDataType.push_back({"global", lexemesList[i + 1], lexemesList[i - 3]});
         } else {
            idNamesWithDataType.push_back({scope, lexemesList[i + 1], lexemesList[i - 3]});
         }
         if (lexemesList[i + 4] == "(") {
            idNamesWithType.push_back({scope, lexemesList[i + 1], "func"});
         } else {
            idNamesWithType.push_back({scope, lexemesList[i + 1], "var"});
         }
```

```
    }
}



scopeFlag = 0;
for (size_t i = 0; i < lexemesList.size(); i++) {
    if (lexemesList[i] == "id" && lexemesList[i + 4] == "(") {
        scope = lexemesList[i + 1];
        scopeFlag = 1;
    } else if (lexemesList[i] == "id" && lexemesList[i + 1] == "main" && lexemesList[i + 4] == "(") {
        scope = "main";
        scopeFlag = 1;
    } else if (lexemesList[i] == "}") {
        scopeFlag = 0;
    } else if (scopeFlag == 0) {
        scope = "global";
    }
    if (lexemesList[i] == "id") {
        if (lexemesList[i + 4] == "=" && lexemesList[i + 7] != "id") {
            idNamesWithValue.push_back({scope, lexemesList[i + 1], lexemesList[i + 7]});
        }
    }
}

int sn = 0;
for (size_t i = 0; i < idNamesWithDataType.size(); i++) {
    sn++;
    std::string name = idNamesWithDataType[i][1];
    std::string idType = idNamesWithType[i][2];
```

```
    std::string dtType = idNamesWithDataType[i][2];
    std::string scp = idNamesWithDataType[i][0];
    std::string values = "";
    for (size_t j = 0; j < idNamesWithValue.size(); j++) {
        if (name == idNamesWithValue[j][1] && scp == idNamesWithValue[j][0]) {
            values = idNamesWithValue[j][2];
            break;
        }
    }
    symbolTable.push_back({sn, name, idType, dtType, scp, values});
}

// Function implementations

// Example usage of the functions
 while (true) {
    std::cout << "\nA. Insert\nB. Set Attribute\nC. Clear\nD. Look Up\nE. Display\n"
              << "Enter the mode (A, B, C, D, or E): ";

    char mode;
    std::cin >> mode;

    switch (mode) {
        case 'A':
            insert();
            break;
        case 'B':
            setAttribute();
            break;
```

```
            case 'C':
                freeSymbolTable();
                break;
            case 'D':
                lookup();
                break;
            case 'E':
                display();
                break;
            default:
                std::cout << "Invalid mode selection." << std::endl;
                break;
        }
    }

    return 0;
}
void display() {
    if (symbolTable.empty()) {
        std::cout << "Symbol table is empty." << std::endl;
    } else {
        std::cout << std::setw(10) << "Sl. No." << std::setw(15) << "Name" << std::setw(10) << "ID Type"
<< std::setw(12)
                  << "Data Type" << std::setw(10) << "Scope" << std::setw(10) << "Value" << std::endl;

        for (const auto& entry : symbolTable) {
            std::cout << std::setw(10) << entry.slNo << std::setw(15) << entry.name << std::setw(10) <<
entry.idType
                      << std::setw(12) << entry.dataType << std::setw(10) << entry.scope << std::setw(10) <<
entry.value
```

```
            << std::endl;
        }
    }
    }

    void lookup() {
        std::string name;
        std::cout << "Enter an Identifier's Name: ";
        std::cin >> name;

        std::cout << std::setw(10) << "Sl. No." << std::setw(15) << "Name" << std::setw(10) << "ID Type"
<< std::setw(12)
             << "Data Type" << std::setw(10) << "Scope" << std::setw(10) << "Value" << std::endl;
        bool found = false;
        for (const auto& entry : symbolTable) {
            if (name == entry.name) {
                std::cout << std::setw(10) << entry.slNo << std::setw(15) << entry.name << std::setw(10) <<
entry.idType
                    << std::setw(12) << entry.dataType << std::setw(10) << entry.scope << std::setw(10) <<
entry.value
                    << std::endl;
                found = true;
            }
        }
        if (!found) {
            std::cout << "Identifier not found in the symbol table." << std::endl;
        }
    }

    void freeSymbolTable() {
```

```
    symbolTable.clear();
    std::cout << "Symbol table has been cleared." << std::endl;
}

void setAttribute() {
    std::string name;
    std::cout << "Enter an Identifier's Name: ";
    std::cin >> name;

    bool found = false;
    for (auto& entry : symbolTable) {
        if (name == entry.name) {
            std::string attribute;
            std::cout << "Enter the new attribute: ";
            std::cin >> attribute;
            entry.idType = attribute;
            found = true;
            break;
        }
    }
    if (!found) {
        std::cout << "Identifier not found in the symbol table." << std::endl;
    }
}

void insert() {
    std::string name;
    std::cout << "Enter an Identifier's Name: ";
    std::cin >> name;
```

```
    int sn=symbolTable.size()+1;
    bool found = false;
    for (const auto& entry : symbolTable) {
        if (name == entry.name) {
            found = true;
            break;
        }
    }
    if (found) {
        std::cout << "Identifier already exists in the symbol table." << std::endl;
    } else {
        std::string idType, dataType, scope, value;
        std::cout << "Enter the ID Type: ";
        std::cin >> idType;
        std::cout << "Enter the Data Type: ";
        std::cin >> dataType;
        std::cout << "Enter the Scope: ";
        std::cin >> scope;
        std::cout << "Enter the Value: ";
        std::cin >> value;
        symbolTable.push_back({++sn, name, idType, dataType, scope, value});
    }
}
```