Name : Aadil Mohamed Puthiyaveetil
Reg No : 22BCE2436
Course Title : Compiler Design Lab
Course Code : BCSE307P

LAB ASSIGNMENT-1

Q) Write a program in any language to create a Symbol Table to Insert, Delete, Display all <token, lexeme> pairs from a given Input code Snippet.

Input code snippet also can be of any language.

Sample Input:

```
float f1;
int x, i=0;
while(i<5)
{
if((i+2) == 3)
      break;
else
      print("Value of i =%d", i);
}
```

Sample Output:
Make a Symbol Table entry where following details should be present

Token, lexeme
id, f1
id, x
id, I
key, float
key, int
.....
....
.....

..so on, all token, lexeme pairs

Code

```
#include <iostream>
#include <string>
#include <unordered_map>
#include <vector>
#include <regex>

using namespace std;

// Structure to represent an entry in the Symbol Table
struct SymbolTableEntry {
    string token;
    string lexeme;
};

// Class to represent the Symbol Table
class SymbolTable {
private:
    vector<SymbolTableEntry> table;

public:
    // Function to insert a new entry into the Symbol Table
    void insert(const string &token, const string &lexeme) {
        table.push_back({token, lexeme});
    }

    // Function to delete an entry from the Symbol Table
    void remove(const string &lexeme) {
        table.erase(remove_if(table.begin(), table.end(), [&](const SymbolTableEntry &entry)
{
            return entry.lexeme == lexeme;
        }), table.end());
    }

    // Function to display all entries in the Symbol Table
    void display() const {
        for (const auto &entry : table) {
            cout << entry.token << ", " << entry.lexeme << endl;
        }
    }
};
```

```
// Function to tokenize the input code snippet
void tokenize(const string &code, SymbolTable &symbolTable) {
    // Define regex patterns for different tokens
    regex keywordPattern("\\b(float|int|while|if|else|break|print)\\b");
    regex identifierPattern("\\b[a-zA-Z_][a-zA-Z0-9_]*\\b");
    regex numberPattern("\\b[0-9]+\\b");
    regex stringPattern("\"[^\"]*\"");
    regex operatorPattern("[+\\-*/=<>!]+");

    // Tokenize the input code
    sregex_iterator iter(code.begin(), code.end(), keywordPattern);
    sregex_iterator end;
    while (iter != end) {
        symbolTable.insert("key", iter->str());
        ++iter;
    }

    iter = sregex_iterator(code.begin(), code.end(), identifierPattern);
    while (iter != end) {
        symbolTable.insert("id", iter->str());
        ++iter;
    }

    iter = sregex_iterator(code.begin(), code.end(), numberPattern);
    while (iter != end) {
        symbolTable.insert("num", iter->str());
        ++iter;
    }

    iter = sregex_iterator(code.begin(), code.end(), stringPattern);
    while (iter != end) {
        symbolTable.insert("str", iter->str());
        ++iter;
    }

    iter = sregex_iterator(code.begin(), code.end(), operatorPattern);
    while (iter != end) {
        symbolTable.insert("op", iter->str());
        ++iter;
    }
}

int main() {
    // Sample input code snippet
    string code = R"(float f1;
int x, i=0;
```

```
while(i<5)
{
if((i+2) == 3)
      break;
else
      print("Value of i =%d", i);
})";

    // Create a Symbol Table
    SymbolTable symbolTable;

    // Tokenize the input code and populate the Symbol Table
    tokenize(code, symbolTable);

    // Display the Symbol Table
    symbolTable.display();

    // Example: Insert a new entry
    symbolTable.insert("id", "newVar");
    cout << "\nAfter inserting newVar:\n";
    symbolTable.display();

    // Example: Delete an entry
    symbolTable.remove("i");
    cout << "\nAfter deleting i:\n";
    symbolTable.display();

    return 0;
}
```

Output

```
key, float
key, int
key, while
key, if
key, break
key, else
key, print
id, float
id, f1
id, int
id, x
id, i
id, while
id, i
id, if
id, i
id, break
id, else
id, print
id, Value
id, of
id, i
id, d
id, i
num, 0
num, 5
num, 2
num, 3
str, "Value of i =%d"
op, =
op, <
op, +
op, ==
op, =
```

```
After inserting newVar:
key, float
key, int
key, while
key, if
key, break
key, else
key, print
id, float
id, f1
id, int
id, x
id, i
id, while
id, i
id, if
id, i
id, break
id, else
id, print
id, Value
id, of
id, i
id, d
id, i
num, 0
num, 5
num, 2
num, 3
str, "Value of i =%d"
op, =
op, <
op, +
op, ==
op, =
id, newVar
```

```
After deleting i:
key, float
key, int
key, while
key, if
key, break
key, else
key, print
id, float
id, f1
id, int
id, x
id, while
id, if
id, break
id, else
id, print
id, Value
id, of
id, d
num, 0
num, 5
num, 2
num, 3
str, "Value of i =%d"
op, =
op, <
op, +
op, ==
op, =
id, newVar
```