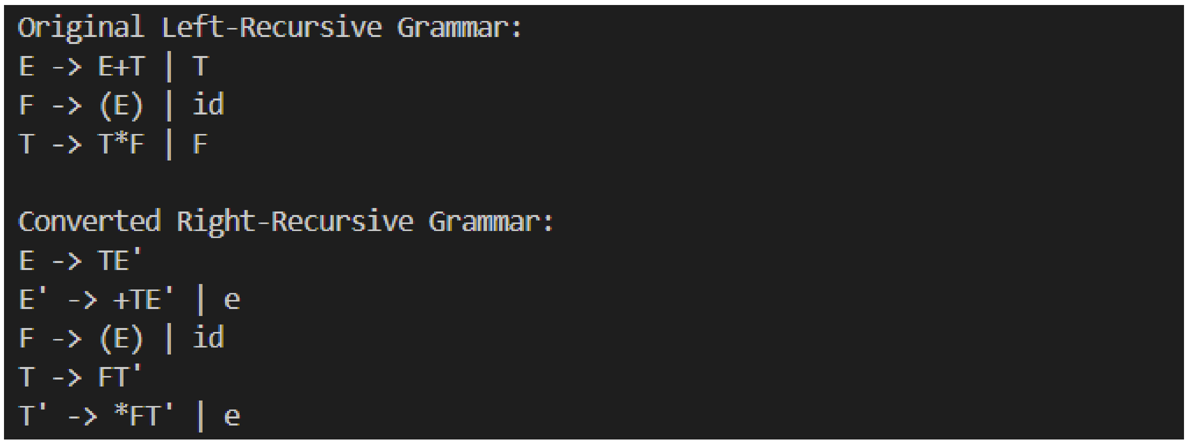
Q7.) Write a program to convert left recursive grammar to right recursive grammar.

```
#include <iostream>
#include <map>
#include <vector>
#include <string>

using namespace std;

//Function to convert left recursive grammar to right recursive grammar
map<string, vector<string>> convertToRightRecursive(const map<string, vector<string>>&
grammar) {
    map<string, vector<string>> newGrammar;
    for (const auto& rule : grammar) {
        string nonTerminal = rule.first;
        vector<string> alpha, beta;

        //Separate left recursive and non-left recursive productions
        for (const auto& production : rule.second) {
            if (production[0] == nonTerminal[0]) {
                //Left recursive production (A -> Aα)
                alpha.push_back(production.substr(1));
            }
            else {
                //Non-left recursive production (A -> β)
                beta.push_back(production);
            }
        }
        if(!alpha.empty()) {
            //There is left recursion to eliminate
            string newNonTerminal = nonTerminal + "'";
            // A -> βA'
            for (const auto& b : beta) {
                newGrammar[nonTerminal].push_back(b + newNonTerminal);
            }
            // A' -> αA' | ε
            for (const auto& a : alpha) {
                newGrammar[newNonTerminal].push_back(a + newNonTerminal);
            }
            newGrammar[newNonTerminal].push_back("e");  //e represents an empty string
        }
        else {
            //No left recursion, keep the rule as is
            newGrammar[nonTerminal] = rule.second;
        }
    }

    return newGrammar;
```

```cpp
}
// Function to print the grammar
void printGrammar(const map<string, vector<string>>& grammar) {
    for (const auto& rule : grammar) {
        cout<<rule.first<<" -> ";
        for (size_t i = 0; i < rule.second.size(); ++i) {
            cout<<rule.second[i];
            if (i < rule.second.size() - 1) cout<<" | ";
        }
        cout<<endl;
    }
}

int main() {
    //Example left-recursive grammar
    map<string, vector<string>> grammar = {
        {"E", {"E+T", "T"}},
        {"T", {"T*F", "F"}},
        {"F", {"(E)", "id"}}
    };

    cout<<"Original Left-Recursive Grammar:\n";
    printGrammar(grammar);

    auto rightRecursiveGrammar = convertToRightRecursive(grammar);

    cout<<"\nConverted Right-Recursive Grammar:\n";
    printGrammar(rightRecursiveGrammar);

    return 0;
}
```

Output)



```
Original Left-Recursive Grammar:
E -> E+T | T
F -> (E) | id
T -> T*F | F

Converted Right-Recursive Grammar:
E -> TE'
E' -> +TE' | e
F -> (E) | id
T -> FT'
T' -> *FT' | e
```