Q6.) Write a program to left factor the given grammar.

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

using namespace std;

//Function to left factor a given grammar
map<string, vector<string>> leftFactorGrammar(const map<string, vector<string>>& grammar) {
    map<string, vector<string>> factoredGrammar;
    bool changed = false;

    for(const auto& rule : grammar) {
        map<string, set<string>> prefixes;
        for (const auto& production : rule.second) {
            string prefix = production.substr(0, 1);
            prefixes[prefix].insert(production);
        }

        for(const auto& prefix : prefixes) {
            if(prefix.second.size() > 1) {
                changed = true;
                string newNonTerminal = rule.first + "'";
                int counter = 1;
                while(factoredGrammar.find(newNonTerminal) != factoredGrammar.end()) {
                    newNonTerminal = rule.first + "'" + to_string(counter++);
                }

                factoredGrammar[rule.first].push_back(prefix.first + newNonTerminal);

                for(const auto& prod : prefix.second) {
                    factoredGrammar[newNonTerminal].push_back(prod.length() > 1 ?
prod.substr(1) : "");
                }
            }
            else factoredGrammar[rule.first].push_back(*prefix.second.begin());
        }
    }

    return changed ? leftFactorGrammar(factoredGrammar) : factoredGrammar;
}

//Function to print the grammar
void printGrammar(const map<string, vector<string>>& grammar) {
    for (auto& rule : grammar){
```

```
        cout<<rule.first<<" -> ";
        for(int i = 0; i < rule.second.size(); ++i){
            cout << rule.second[i];
            if(i < rule.second.size() - 1) cout<<" | ";
        }
        cout<<endl;
    }
}

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

    cout<<"Original Grammer:\n";
    printGrammar(grammar);

    auto factoredGrammar = leftFactorGrammar(grammar);

    cout<<"\nLeft Factored Grammar:\n";
    printGrammar(factoredGrammar);

    return 0;
}
```

Output)

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

Left Factored Grammar:
E -> EE'
E' -> +E''
E'' -> TE'''
E''' ->  | +T
F -> (E) | id
T -> FT'
T' -> *T''
T'' -> F | T
```