Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Course Code: BCSE307P

Course Name: Compiler Design Lab

Assessment – 3

Name: Shivam Dave

Reg. No.: 21BCB0107

Write a program in c/c++

3a) Compute first and follow and Construct predictive parsing table.

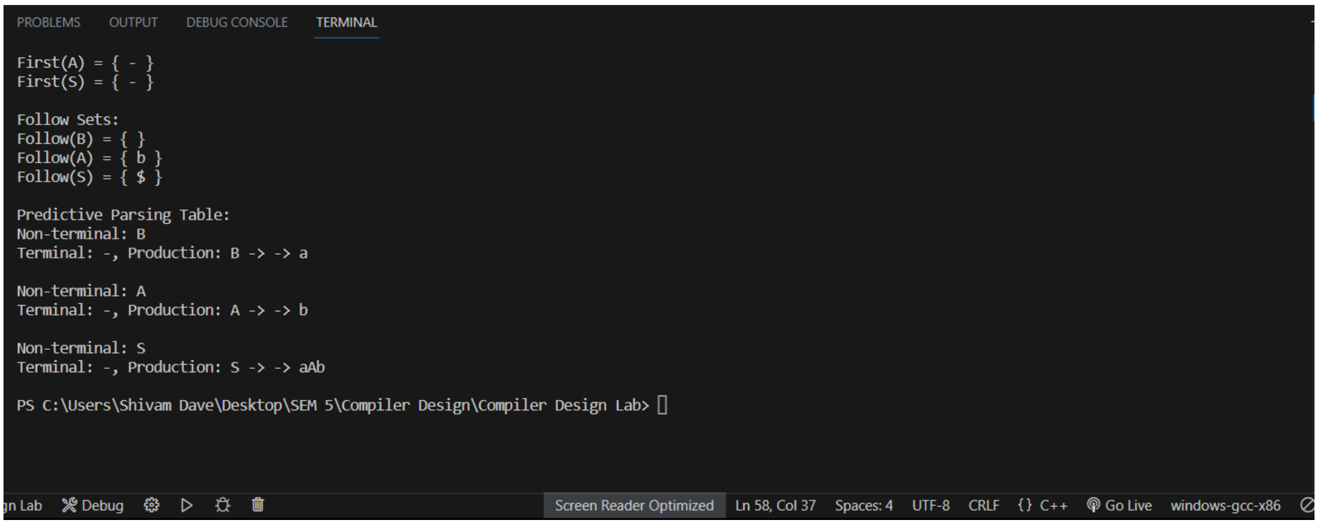
Code:

```cpp
#include <iostream>
#include <unordered_map>
#include <unordered_set>
#include <vector>

using namespace std;

struct Production {
    char lhs;
    string rhs;
};

unordered_map<char, unordered_set<char>> firstSet;
unordered_map<char, unordered_set<char>> followSet;
unordered_map<char, unordered_map<char, Production>> parsingTable;

void computeFirst(char symbol, const vector<Production>& productions);
void computeFollow(char symbol, const vector<Production>& productions);
void computePredictiveParsingTable(const vector<Production>& productions);

void printFirstSets();
void printFollowSets();
void printParsingTable();

int main() {
    int numProductions;
    cout << "Enter the number of productions: ";
    cin >> numProductions;

    vector<Production> productions(numProductions);

    for (int i = 0; i < numProductions; ++i) {
        cout << "Enter production " << (i + 1) << " (in the form A -> alpha): 
";
        cin >> productions[i].lhs;
        cin.ignore();  // Ignore newline character
        getline(cin, productions[i].rhs);
    }
    for (const Production& production : productions) {
        computeFirst(production.lhs, productions);
        computeFollow(production.lhs, productions);
    }

    computePredictiveParsingTable(productions);

    cout << "\nFirst Sets:\n";
    printFirstSets();

    cout << "\nFollow Sets:\n";
    printFollowSets();

    cout << "\nPredictive Parsing Table:\n";
    printParsingTable();

    return 0;
}

void computeFirst(char symbol, const vector<Production>& productions) {
    if (firstSet.find(symbol) != firstSet.end()) {
        return;
    }

    unordered_set<char>& first = firstSet[symbol];

    for (const Production& production : productions) {
        if (production.lhs == symbol) {
            char firstSymbol = production.rhs[0];

            if (isupper(firstSymbol)) {
                computeFirst(firstSymbol, productions);

                unordered_set<char>& firstOfFirstSymbol =
firstSet[firstSymbol];
                first.insert(firstOfFirstSymbol.begin(),
firstOfFirstSymbol.end());
            } else {
                first.insert(firstSymbol);
            }
        }
    }
}

void computeFollow(char symbol, const vector<Production>& productions) {
    if (followSet.find(symbol) != followSet.end()) {
        return;
```

```cpp
    }

    unordered_set<char>& follow = followSet[symbol];

    if (symbol == productions[0].lhs) {
        follow.insert('$');
    }

    for (const Production& production : productions) {
        const string& rhs = production.rhs;

        size_t pos = rhs.find(symbol);
        while (pos != string::npos) {
            if (pos < rhs.size() - 1) {
                char nextSymbol = rhs[pos + 1];

                if (isupper(nextSymbol)) {
                    unordered_set<char>& firstOfNextSymbol =
firstSet[nextSymbol];
                    follow.insert(firstOfNextSymbol.begin(),
firstOfNextSymbol.end());

                    if (firstOfNextSymbol.find('#') !=
firstOfNextSymbol.end()) {
                        follow.erase('#');
                        pos++;
                    } else {
                        break;
                    }
                } else {
                    follow.insert(nextSymbol);
                    break;
                }
            } else {
                if (production.lhs != symbol) {
                    computeFollow(production.lhs, productions);
                    unordered_set<char>& followOfLHS =
followSet[production.lhs];
                    follow.insert(followOfLHS.begin(), followOfLHS.end());
                }

                break;
            }

            pos = rhs.find(symbol, pos + 1);
        }
    }
}
```

```cpp
void computePredictiveParsingTable(const vector<Production>& productions) {
    for (const Production& production : productions) {
        char lhs = production.lhs;
        const string& rhs = production.rhs;

        char firstSymbol = rhs[0];
        if (isupper(firstSymbol)) {
            unordered_set<char>& first = firstSet[firstSymbol];
            for (char terminal : first) {
                if (terminal != '#') {
                    parsingTable[lhs][terminal] = production;
                }
            }

            if (first.find('#') != first.end()) {
                unordered_set<char>& follow = followSet[lhs];
                for (char terminal : follow) {
                    parsingTable[lhs][terminal] = production;
                }
            }
        } else {
            parsingTable[lhs][firstSymbol] = production;
        }
    }
}

void printFirstSets() {
    for (const auto& entry : firstSet) {
        char symbol = entry.first;
        const unordered_set<char>& first = entry.second;

        cout << "First(" << symbol << ") = { ";
        for (char ch : first) {
            cout << ch << " ";
        }
        cout << "}\n";
    }
}

void printFollowSets() {
    for (const auto& entry : followSet) {
        char symbol = entry.first;
        const unordered_set<char>& follow = entry.second;

        cout << "Follow(" << symbol << ") = { ";
        for (char ch : follow) {
            cout << ch << " ";
        }
        cout << "}\n";
    }
}

void printParsingTable() {
    for (const auto& row : parsingTable) {
        char lhs = row.first;
        const unordered_map<char, Production>& tableRow = row.second;

        cout << "Non-terminal: " << lhs << "\n";

        for (const auto& entry : tableRow) {
            char terminal = entry.first;
            const Production& production = entry.second;

            cout << "Terminal: " << terminal << ", Production: " <<
production.lhs << " -> " << production.rhs << "\n";
        }

        cout << "\n";
    }
}
```

OUTPUT:



```
First(A) = { - }
First(S) = { - }

Follow Sets:
Follow(B) = { }
Follow(A) = { b }
Follow(S) = { $ }

Predictive Parsing Table:
Non-terminal: B
Terminal: -, Production: B -> -> a

Non-terminal: A
Terminal: -, Production: A -> -> b

Non-terminal: S
Terminal: -, Production: S -> -> aAb

PS C:\Users\Shivam Dave\Desktop\SEM 5\Compiler Design\Compiler Design Lab> []
```

b) Check whether the given grammar is LL(1) or not?

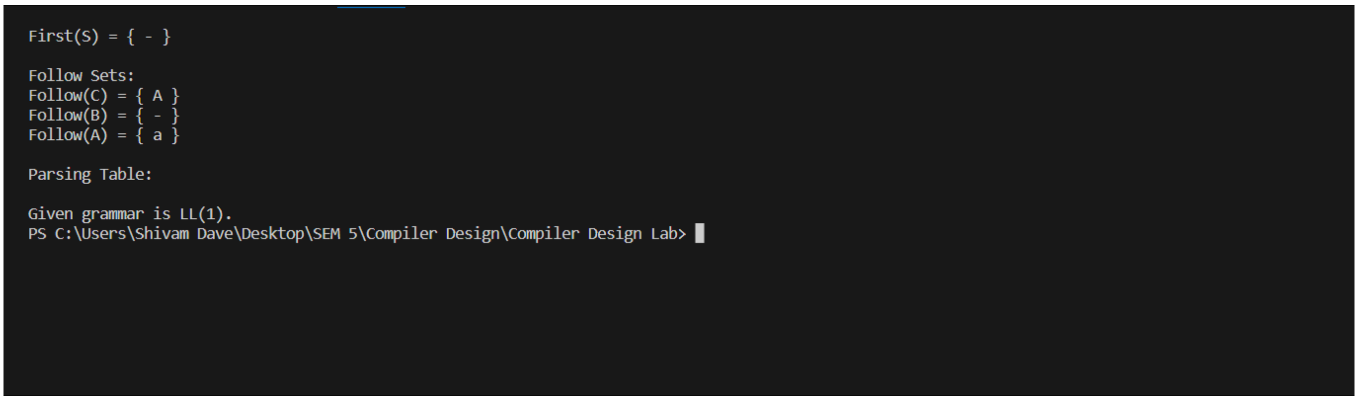
Code:

```
#include <iostream>
#include <unordered_map>
#include <unordered_set>
#include <vector>
using namespace std;

struct Production {
    char lhs;
    string rhs;
};

unordered_map<char, unordered_set<char>> firstSet;
unordered_map<char, unordered_set<char>> followSet;
unordered_map<char, unordered_map<char, Production>> parsingTable;

void computeFirstSets(const vector<Production>& productions);
void computeFollowSets(const vector<Production>& productions);
bool isLL1Grammar(const vector<Production>& productions);
void printFirstSets();
void printFollowSets();
void printParsingTable();

int main() {
    int numProductions;
    cout << "Enter the number of productions: ";
    cin >> numProductions;

    vector<Production> productions(numProductions);
    cout << "Enter the productions in the form A -> alpha:\n";
    for (int i = 0; i < numProductions; i++) {
        cout << "Enter production " << i + 1 << ": ";
        cin >> productions[i].lhs;
        cin.ignore(); // Ignore the '->' symbol
        getline(cin, productions[i].rhs);
    }

    computeFirstSets(productions);
    computeFollowSets(productions);
    bool isLL1 = isLL1Grammar(productions);

    cout << "\nFirst Sets:\n";
    printFirstSets();

    cout << "\nFollow Sets:\n";
    printFollowSets();

    cout << "\nParsing Table:\n";
    printParsingTable();

    cout << "\nGiven grammar is" << (isLL1 ? " LL(1)." : " not LL(1).") <<
endl;

    return 0;
}

void computeFirstSets(const vector<Production>& productions) {
    for (const Production& production : productions) {
        char lhs = production.lhs;
        const string& rhs = production.rhs;

        char firstSymbol = rhs[0];
        if (isupper(firstSymbol)) {
            unordered_set<char>& first = firstSet[lhs];
            first.insert(firstSymbol);
        } else {
            unordered_set<char>& first = firstSet[lhs];
            first.insert(firstSymbol);
        }
    }
}

void computeFollowSets(const vector<Production>& productions) {
    for (const Production& production : productions) {
        char lhs = production.lhs;
        const string& rhs = production.rhs;

        for (int i = 0; i < rhs.length(); i++) {
            char symbol = rhs[i];
            if (isupper(symbol)) {
                unordered_set<char>& follow = followSet[symbol];
                if (i == rhs.length() - 1) {
                    follow.insert(lhs);
                } else {
                    char nextSymbol = rhs[i + 1];
                    if (isupper(nextSymbol)) {
                        const unordered_set<char>& first =
firstSet[nextSymbol];
                        follow.insert(first.begin(), first.end());
                    } else {
                        follow.insert(nextSymbol);
                    }
                }
```

```cpp
            }
        }
    }
}

bool isLL1Grammar(const vector<Production>& productions) {
    for (const Production& production : productions) {
        char lhs = production.lhs;
        const string& rhs = production.rhs;

        char firstSymbol = rhs[0];
        if (!isupper(firstSymbol)) {
            continue;
        }

        unordered_map<char, Production>& tableRow = parsingTable[lhs];
        const unordered_set<char>& first = firstSet[firstSymbol];
        for (char terminal : first) {
            if (tableRow.find(terminal) != tableRow.end()) {
                return false;
            }
            tableRow[terminal] = production;
        }

        if (first.find('#') != first.end()) {
            const unordered_set<char>& follow = followSet[lhs];
            for (char terminal : follow) {
                if (tableRow.find(terminal) != tableRow.end()) {
                    return false;
                }
                tableRow[terminal] = production;
            }
        }
    }

    return true;
}

void printFirstSets() {
    for (const auto& entry : firstSet) {
        cout << "First(" << entry.first << ") = { ";
        const unordered_set<char>& first = entry.second;
        for (char symbol : first) {
            cout << symbol << " ";
        }
        cout << "}\n";
    }
}
```

```cpp
void printFollowSets() {
    for (const auto& entry : followSet) {
        cout << "Follow(" << entry.first << ") = { ";
        const unordered_set<char>& follow = entry.second;
        for (char symbol : follow) {
            cout << symbol << " ";
        }
        cout << "}\n";
    }
}

void printParsingTable() {
    for (const auto& entry : parsingTable) {
        char nonTerminal = entry.first;s
        const unordered_map<char, Production>& tableRow = entry.second;

        for (const auto& cell : tableRow) {
            char terminal = cell.first;
            const Production& production = cell.second;

            cout << "M[" << nonTerminal << ", " << terminal << "] = ";
            cout << production.lhs << " -> " << production.rhs << endl;
        }
    }
}
```

Output:



```
First(S) = { - }

Follow Sets:
Follow(C) = { A }
Follow(B) = { - }
Follow(A) = { a }

Parsing Table:

Given grammar is LL(1).
PS C:\Users\Shivam Dave\Desktop\SEM 5\Compiler Design\Compiler Design Lab>
```