Practical 5\LL1.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

void first(map<char, vector<string>> &grammar, char ch, set<char> &F, char epsilon)
{
    for (const auto &rule : grammar[ch])
    {
        char f = rule[0];
        if (isupper(f))
        {
            set<char> tempFirst;
            first(grammar, f, tempFirst, epsilon);
            F.insert(tempFirst.begin(), tempFirst.end());
            if (tempFirst.find(epsilon) != tempFirst.end() && rule.length() > 1)
            {
                first(grammar, rule[1], F, epsilon);
            }
        }
        else
        {
            F.insert(f);
        }
    }
}

void follow(map<char, vector<string>> &grammar, char ch, set<char> &F, char startSymbol, set<char> &visited, char epsilon)
{
    if (visited.find(ch) != visited.end())
        return;
    visited.insert(ch);

    if (ch == startSymbol)
        F.insert('$');

    for (auto &rule : grammar)
    {
        for (const auto &prod : rule.second)
        {
            for (size_t i = 0; i < prod.size(); i++)
            {
                if (prod[i] == ch)
                {
                    if (i + 1 < prod.size())
                    {
                        char nextChar = prod[i + 1];
                        if (isupper(nextChar))
                        {
                            set<char> tempFirst;
                            first(grammar, nextChar, tempFirst, epsilon);
                            F.insert(tempFirst.begin(), tempFirst.end());
                            F.erase(epsilon);

                            if (tempFirst.find(epsilon) != tempFirst.end())
```

```
                                follow(grammar, rule.first, F, startSymbol, visited, epsilon);
                        }
                        else
                            F.insert(nextChar);
                    }
                    else
                    {
                        if (rule.first != ch)
                            follow(grammar, rule.first, F, startSymbol, visited, epsilon);
                    }
                }
            }
        }
    }
}

void printTable(map<char, map<char, string>> &table, vector<char> &terminals, vector<char> &nonTerminals)
{
    cout << "\nLL(1) Parsing Table:\n";
    cout << setw(6) << " ";

    for (auto &t : terminals)
        cout << setw(6) << t;
    cout << endl;

    for (auto &nt : nonTerminals)
    {
        cout << setw(6) << nt;
        for (auto &t : terminals)
        {
            if (table[nt].find(t) != table[nt].end())
                cout << setw(6) << table[nt][t];
            else
                cout << setw(6) << " ";
        }
        cout << endl;
    }
}

int main()
{
    map<char, vector<string>> grammar;
    int ruleCount;
    char startSymbol, epsilon;

    cout << "Enter the epsilon character (e.g. '#' or 'e' for empty productions): ";
    cin >> epsilon;

    cout << "How many grammar rules: ";
    cin >> ruleCount;

    cout << "Enter the start symbol: ";
    cin >> startSymbol;
```

```
107
108     cout << "Enter the grammar (e.g. E→E+T|T):\n";
109     for (int i = 0; i < ruleCount; i++)
110     {
111         char nonTerminal;
112         string production;
113
114         cout << "Enter non-terminal: ";
115         cin >> nonTerminal;
116         cout << nonTerminal << "→";
117         cin >> production;
118
119         stringstream ss(production);
120         string prod;
121         while (getline(ss, prod, '|'))
122             grammar[nonTerminal].push_back(prod);
123     }
124
125     set<char> terminalsSet, nonTerminalsSet;
126
127     for (auto &rule : grammar)
128     {
129         nonTerminalsSet.insert(rule.first);
130         for (const auto &prod : rule.second)
131         {
132             for (char ch : prod)
133             {
134                 if (!isupper(ch) && ch != epsilon)
135                     terminalsSet.insert(ch);
136             }
137         }
138     }
139     terminalsSet.insert('$');
140
141     vector<char> terminals(terminalsSet.begin(), terminalsSet.end());
142     vector<char> nonTerminals(nonTerminalsSet.begin(), nonTerminalsSet.end());
143
144     map<char, map<char, string>> table;
145
146     for (auto &rule : grammar)
147     {
148         char nonTerminal = rule.first;
149         for (const auto &prod : rule.second)
150         {
151             set<char> firstSet;
152             if (isupper(prod[0]))
153                 first(grammar, prod[0], firstSet, epsilon);
154             else
155                 firstSet.insert(prod[0]);
156
157             for (char terminal : firstSet)
158             {
159                 if (terminal != epsilon)
160                     table[nonTerminal][terminal] = prod;
161                 else
162                 {
163                     set<char> followSet;
```

```
                    set<char> visited;
                    follow(grammar, nonTerminal, followSet, startSymbol,
visited, epsilon);
                    for (char followChar : followSet)
                        table[nonTerminal][followChar] = prod;
                    if (followSet.find('$') == followSet.end())
                        table[nonTerminal]['$'] = prod;
                }
            }
        }
    }

    printTable(table, terminals, nonTerminals);

    return 0;
}

```