Construct the LL(1) Parsing table for a CFG given through a file
Lab 6
Aiman Fatima
20BCS008

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



#include <vector>
#include <string>
#include <deque>
#include <sstream>
#include <regex>
#include <iomanip>

using namespace std;
multimap<string, deque<string>> m;
map<string, bool> Noterm;
set<string> Term;
map<string, int> PosTerm, PosNoTerm;
map<string, vector<string>> First;
map<string, set<string>> Follow;
vector<string> string_test;
void ReadGrammar()
{
    string s, flecha;
    string k = "@", ini;
    while (getline(cin, s))
    {
        if (s == "string_test:")
            break;
        stringstream in(s);
        in >> ini >> flecha;
        if (k == "@")
            Noterm[ini] = 1;
        else
            Noterm[ini] = 0;
        deque<string> valores;
        while (in >> k)
        {
            if (k == "|")
                m.insert({ini, valores}), valores.clear();
            else
            {
                bool ok = 1;
                for (auto ch : k)
                    if (ch >= 'A' && ch <= 'Z')
                        ok = 0;
                valores.push_back(k);
                if (ok)
                    Term.insert(k);
            }
        }
        m.insert({ini, valores});
    }
    getline(cin, s);
    stringstream in(s);
    string_test = {};
    while (in >> k)
        string_test.push_back(k);
    string_test.push_back("$");
    reverse(string_test.begin(), string_test.end());
}
void Recursion()
{
    multimap<string, deque<string>> New;
    set<string> NewNoterm;
    for (auto e : Noterm)
    {
        bool ok = 0;
        for (auto val : m)
            if (val.first == e.first && e.first == val.second.front())
                ok = 1;
        if (ok)
        {
            string ini = e.first + "'";
            while (Noterm.find(ini) != Noterm.end())
                ini += "'";
            NewNoterm.insert(ini);
            deque<string> d;
            for (auto val : m)
            {
                if (val.first == e.first)
                {
                    d = val.second;
                    if (e.first != d.front())
                    {
                        d.push_back(ini);
                        New.insert({e.first, d});
                    }
                    else
                    {
                        d.pop_front();
                        d.push_back(ini);
                        New.insert({ini, d});
                    }
                }
            }
            d = {"Ɛ"};
            New.insert({ini, d});
        }
        else
        {
            for (auto val : m)
                if (val.first == e.first)
                    New.insert(val);
        }
    }
    for (auto e : NewNoterm)
```

```
        Noterm[e] = 0;
    m = New;
}
void Ambiguity()
{
    multimap<string, deque<string>> New;
    set<string> NewNoterm;
    for (auto e : Noterm)
    {
        map<string, int> cnt;
        for (auto val : m)
            if (val.first == e.first)
            {
                cnt[val.second.front()]++;
            }
        int mx = 0;
        string mxs;
        for (auto ele : cnt)
            if (ele.second > mx)
                mx = ele.second, mxs = ele.first;
        if (mx <= 1)
        {
            for (auto val : m)
                if (val.first == e.first)
                    New.insert(val);
            continue;
        }
        string ini = e.first + "'";
        while (Noterm.find(ini) != Noterm.end())
            ini += "'";
        NewNoterm.insert(ini);
        deque<string> d;
        for (auto val : m)
        {
            if (val.first == e.first)
            {
                d = val.second;
                if (mxs == d.front())
                {
                    d.pop_front();
                    if (!d.size())
                        d = {"Ɛ"};
                    New.insert({ini, d});
                }
                else
                    New.insert({e.first, d});
            }
        }
        d = {mxs, ini};
        New.insert({e.first, d});
    }
    for (auto e : NewNoterm)
        Noterm[e] = 0;
    m = New;
}
map<string, int> vis;
vector<string> DfsFirst(string e)
```

```
{
    vis[e] = 1;
    if (!m.count(e))
    {
        First[e] = {e};
        return {e};
    }
    vector<string> res;
    for (auto val : m)
    {
        if (val.first == e)
        {
            vector<string> ter;
            ter = DfsFirst(val.second.front());
            for (auto u : ter)
                res.push_back(u);
        }
    }
    First[e] = res;
    return res;
}
void CalcFirst()
{
    for (auto e : Term)
        First[e] = {e};
    for (auto e : Noterm)
    {
        if (!vis[e.first])
            First[e.first] = DfsFirst(e.first);
    }
}
map<string, int> visg;
void DfsFollow(string e)
{
    map<string, int> used;
    vector<string> st;
    st.push_back(e);
    while (st.size())
    {
        e = st.back();
        if (visg[e])
        {
            st.pop_back();
            if (st.size())
            {
                string v = st.back();
                for (auto ele : Follow[e])
                    Follow[v].insert(ele);
            }
            used[e] = 0;
            continue;
        }
        visg[e] = used[e] = 1;
        for (auto val : m)
        {
            deque<string> d = val.second;
            for (int i = 0; i < d.size(); i++)
            {
                if (e == d[i])
                {
                    bool ok = 0;
                    if (i + 1 < d.size())
                    {
                        vector<string> res = First[d[i + 1]];
                        for (auto v : res)
                        {
                            if (v == "Ɛ")
                                ok = 1;
                            else
                                Follow[e].insert(v);
                        }
                    }
                    if (i + 1 >= d.size() || ok)
                    {
                        if (used[val.first])
                        {
                            for (int j = 0; j < st.size(); j++)
                            {
                                if (st[j] == val.first)
                                {
                                    st.insert(st.begin() + j, e);
                                    break;
                                }
                            }
                        }
                        else
                            st.push_back(val.first);
                    }
                }
            }
        }
    }
}
void CalcFollow()
{
    for (auto e : Noterm)
    {
        if (e.second)
            Follow[e.first].insert("$");
        if (!visg[e.first])
            DfsFollow(e.first);
    }
}
deque<string> table[22][22];
void TableLL()
{
    for (auto e : First)
    {
        for (auto v : e.second)
        {
            deque<string> d;
            if (v == "Ɛ")
            {
                d = {"Ɛ"};
                for (auto val : Follow[e.first])
                {
                    table[PosNoTerm[e.first]][PosTerm[val]] = d;
                }
            }
            else
            {
                for (auto val : m)
                {
                    if (val.first == e.first)
                    {
                        bool ok = 0;
                        for (auto u : First[val.second.front()])
                        {
                            if (v == u)
                                ok = 1;
                        }
                        if (ok)
                        {
                            d = val.second;
                            table[PosNoTerm[e.first]][PosTerm[v]] = d;
                            break;
                        }
                    }
                }
            }
        }
    }
}
void valid()
{
    string ini;
    for (auto e : Noterm)
        if (e.second)
            ini = e.first;
    vector<string> pila;
    pila.push_back("$");
    pila.push_back(ini);
    bool ok = 1;
    string line;
    line = "stack";
    cout << line << string(20 - line.size(), ' ');
    line = "string";
    cout << string(30 - line.size(), ' ') << line;
    line = "Action";
    cout << string(40 - line.size(), ' ') << line;
    cout << endl;
    while (pila.size() && string_test.size())
    {
        line = "";
        for (int i = pila.size() - 1; i >= 0; i--)
            line += pila[i] + " ";
        cout << line << string(20 - line.size(), ' ');
        if (!ok)
        {
            break;
        }
        line = "";
        for (int i = string_test.size() - 1; i >= 0; i--)
            line += string_test[i] + " ";
        cout << string(30 - line.size(), ' ') << line;

        auto u = pila.back();
        pila.pop_back();
        if (u == string_test.back() || u == "Ɛ")
        {
            line = u;
            if (u == string_test.back())
            {
                string_test.pop_back();
                if (!string_test.size())
                    line = "Accepted";
                else
                    line = "match";
            }
            cout << string(40 - line.size(), ' ') << line;
        }
        else
        {
            deque<string> d;
            d = table[PosNoTerm[u]][PosTerm[string_test.back()]];
            line = "";
            for (auto e : d)
                line += e;
            reverse(d.begin(), d.end());
            for (auto e : d)
            {
                if (e == "error")
                    ok = 0;
                pila.push_back(e);
            }
            cout << string(40 - line.size(), ' ') << line;
        }
        cout << endl;
        if (!ok || !string_test.size())
            break;
    }
}
void validShowGrammar()
{
    Ambiguity();
    Recursion();
    cout << "Rules after resolving ambiguity and Left Recursion: \n";
    for (auto e : m)
    {
        cout << e.first << " -> ";
        for (auto val : e.second)
            cout << val << " ";
        cout << endl;
    }
    cout << endl;
}
void ShowFirst()
{
    cout << "First: \n";
    for (auto e : First)
    {
        cout << e.first << " -> ";
        for (auto v : First[e.first])
            cout << v << " ";
        cout << endl;
    }
    cout << endl;
}
void ShowFollow()
{
    cout << "Follow: \n";
    for (auto e : Follow)
    {
        cout << e.first << " -> ";
        for (auto v : Follow[e.first])
            cout << v << " ";
        cout << endl;
    }
    cout << endl;
}
void ShowTable()
{
    cout << "LL Analyzer Table:\n";
    int w = 9;
    cout << string(2, ' ');
    for (auto v : Term)
    {
        int l = (w - v.size()) / 2;
        int r = w - l - v.size();
        cout << "|" << string(l, ' ') << v << string(r, ' ');
    }
    cout << endl;
    for (auto v : Term)
        cout << string(w + 1, '-');
    cout << endl;
    for (auto u : First)
    {
        if (!m.count(u.first))
            continue;
        cout << setw(2) << left << u.first;
        for (auto v : Term)
        {
            string line;
            for (auto e : table[PosNoTerm[u.first]][PosTerm[v]])
                line += e + " ";
            int l = (w - (int)line.size()) / 2;
            int r = w - l - line.size();
            if (line == "Ɛ ")
                r++;
            cout << "|" << string(l, ' ') << line << string(r, ' ');
        }
        cout << endl;
    }
    for (auto v : Term)
        cout << string(w + 1, '-');
    cout << "\n\n";
}
int main()
{
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    ReadGrammar();
    validShowGrammar();
    CalcFirst();
    CalcFollow();
    Term.insert("$");
    int cntTerm = 0;
    for (auto e : Term)
        PosTerm[e] = cntTerm++;
    int cntNoTerm = 0;
    for (auto e : Noterm)
        PosNoTerm[e.first] = cntNoTerm++;
    /*  Show First*/
    ShowFirst();
    /*  Show Follow*/
    ShowFollow();
    for (int i = 0; i < cntNoTerm; i++)
        for (int j = 0; j < cntTerm; j++)
        {
            table[i][j] = {"error"};
        }
    TableLL();
    // Show Table///
    ShowTable();
    valid();
}
```