LFA pentru Gabi

```
/**
        @author RDP

        Full LL1 PARSER

                +++ LIMITATIONS   +++
                        1.We need to enter grammer after removing recursion and doing
left factoring.
                        2.All symbols must be of length 1. (e.x. E' is not allowed).
                                however you can use e instead of E'.
                                (another e.x. id is not allowed - You can use i instead of id).
                        3. Here $ = epselon and & =$ (Please keep this in mind while
giving input and observing output).
                        4. If the grammer is wrong program may go in infinte loop.
                        5. Space is required around "->" and "|" .
                                However you can ignore spaces in (,),+,*,/,- etc..
                        6. In any input rule there shouldn't be any space.
                                {
                                For e.x.
                                        A -> ab | d       -> RIGHT
                                        A -> a b | d      -> WRONG
                                }
                        7.Violation of any of these rules can lead to undefined behaviour.
                        8...//
;
**/
#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <sstream>
#include <algorithm>
using namespace std;
map <string,string> first,follow,rules;
set <string> nt,t;
vector <string> calls_for_nt;
map < pair<string,string> , string > parse_table;
vector <string> stack,cur_str,rule_used;
string to_string(char c)
{
    string s="";
    s+=c;
    return s;
```

```
}
void set_map(string s)
{
    stringstream ss(s);
    string key,value,temp;
    ss>>key;
    calls_for_nt.push_back(key);
    while(ss>>temp)
    {
        if(temp!="->"&&temp!="|")
            value+=" "+temp;
    }
    rules[key]=value;
    return ;
}
bool is_nterminal(string s)
{
    if(find(nt.begin(),nt.end(),s)!=nt.end())
        return true;
    else
        return false;
}
bool is_terminal(string s)
{
    if(find(t.begin(),t.end(),s)!=t.end())
        return true;
    else
        return false;
}
void set_first(string s)
{
    if(first[s].length()!=0)
        return ;
    string temp=rules[s].substr(1,rules[s].length()-1);
    stringstream ss(temp);
    while(ss>>temp)
    {
        if(is_nterminal(to_string(temp[0])))
        {
            set_first(to_string(temp[0]));
            first[s]=first[to_string(temp[0])];
        }
        else
            first[s]+=temp.substr(0,1)+" ";
    }
    return;
```

```
}
bool check_dol(string s)
{
    for(int i=0;i<s.length();i++)
        if(s[i]=='$')
            return true;
    return false;
}
void set_follow(string s)
{
    int flag=0;
    for(auto itr=rules.begin();itr!=rules.end();itr++)
    {
        stringstream ss(itr->second);
        string temp;
        while(ss>>temp)
        {
            for(int i=0;i<temp.length();i++)
            {
                flag=0;
                if(temp[i]==s[0])
                {
                    if(i+1<temp.length())
                    {
                        if(is_terminal(to_string(temp[i+1])))
                        {
                            follow[s]+=""+to_string(temp[i+1])+"";
                        }
                        else
                        {
                            if(check_dol(rules[to_string(temp[i+1])]))
                            {
                                for(int j=0;j<first[to_string(temp[i+1])].length();j++)
                                    if(first[to_string(temp[i+1])][j]!='$')
                                        follow[s]+=""+to_string(first[to_string(temp[i+1])][j]);
                                follow[s]+=" ";
                            }
                            follow[s]+=""+follow[to_string(temp[i+1])];
                        }
                    }
                    else
                    {
                        follow[s]+=""+follow[itr->first]+" ";
                    }
                    flag=1;
                    break;
                }
            }
            if(flag==1)
                break;
        }
    }
}
string in_rules(string s, string t)
{
    if(find(rules[s].begin(),rules[s].end(),t[0])!=rules[s].end())
    {
        stringstream ss(rules[s]);
        string temp;
        while(ss>>temp)
            if(find(temp.begin(),temp.end(),t[0])!=temp.end())
                return temp;
    }
    else
    {
        stringstream ss(rules[s]);
        string temp;
        ss>>temp;
        return temp;
    }
    return t;
}
void set_parse_table(string s)
{
    string temp=first[s];
    if(find(temp.begin(),temp.end(),'$')!=temp.end())
    {
        string for_dol;
        stringstream ss(follow[s]);
        while(ss>>for_dol)
        {
            parse_table[{s,for_dol}]=s+"-> $";
        }
    }
    stringstream ss(temp);
    while(ss>>temp)
    {
        if(temp==to_string('$'))
            continue;
        parse_table[{s,temp}]=s+"-> "+in_rules(s,temp);
    }
    return ;
```

```
}
void check_str()
{
    if(stack[stack.size()-1][0]=='&' && cur_str[cur_str.size()-1][0]=='&')
        return ;
    if(stack[stack.size()-1][0]==cur_str[cur_str.size()-1][0])
    {

stack.push_back(stack[stack.size()-1].substr(1,stack[stack.size()-1].length()-1)
);

cur_str.push_back(cur_str[cur_str.size()-1].substr(1,cur_str[cur_str.size()-1].leng
th()-1));
        rule_used.push_back(" ");
        check_str();
    }
    else if(parse_table[{to_string(stack[stack.size()-1]
[0]),to_string(cur_str[cur_str.size()-1][0])}].length()!=0)
    {
        stringstream ss(parse_table[{to_string(stack[stack.size()-1]
[0]),to_string(cur_str[cur_str.size()-1][0])}]);
        string temp;
        ss>>temp;ss>>temp;
        if(temp=="$")
        {

stack.push_back(stack[stack.size()-1].substr(1,stack[stack.size()-1].length()-1)
);
            cur_str.push_back(cur_str[cur_str.size()-1]);
            rule_used.push_back(parse_table[{to_string(stack[stack.size()-1]
[0]),to_string(cur_str[cur_str.size()-1][0])}]);
        }
        else
        {

stack.push_back(temp+stack[stack.size()-1].substr(1,stack[stack.size()-1].leng
th()-1));
            cur_str.push_back(cur_str[cur_str.size()-1]);
            rule_used.push_back(parse_table[{to_string(stack[stack.size()-1]
[0]),to_string(cur_str[cur_str.size()-1][0])}]);
        }
        check_str();
    }
    else
    {
        return;
    }
}
int main()
{
    int n;
    cout<<"HOW MANY RULES ARE THERE ? ";
    cin>>n;
    cout<<"ENTER THEM ONE BY ONE : "<<endl;
    string s;
    getline(cin,s);
    for(int i=0;i<n;i++)
    {
        getline(cin,s);
        set_map(s);
    }
    cout<<"MAP IS : \n";
    for(auto itr=rules.begin();itr!=rules.end();itr++)
        cout<<itr->first<<" "<<itr->second<<endl;
    for(auto itr=rules.begin();itr!=rules.end();itr++)
    {
        nt.insert(itr->first);
    }
    for(auto itr=rules.begin();itr!=rules.end();itr++)
    {
        stringstream ss(itr->second);
        string test;
        while(ss>>test)
        {
            for(int i=0;i<test.length();i++)
                if(!is_nterminal(test.substr(i,1))&&(test.substr(i,1))!="$")
                    t.insert(test.substr(i,1));
        }
    }
    cout<<"NON-TERMINALS : "<<endl;
    for(string s:nt)
        cout<<s<<" ";
    cout<<"\nTERMINALS: "<<endl;
    for(string s:t)
        cout<<s<<" ";
    for(string s:nt)
        set_first(s);
    follow.clear();
    for(string s:calls_for_nt)
        set_follow(s);
    for(auto itr=follow.begin();itr!=follow.end();itr++)
        itr->second+=" &";
    cout<<"\nALL FIRST'S : "<<endl;
    for(auto itr=first.begin();itr!=first.end();itr++)
        cout<<itr->first<<" -> "<<itr->second<<endl;
    cout<<"\nALL FOLLOW'S : "<<endl;
    for(auto itr=follow.begin();itr!=follow.end();itr++)
        cout<<itr->first<<" -> "<<itr->second<<endl;
    for(string s:calls_for_nt)
        set_parse_table(s);
    cout<<"\n PARSE TABLE CONTENTS :"<<endl<<endl;
    cout<<"NT\t";
    for(string s:t)
    {
        cout<<s<<"\t";
    }
    cout<<"&\t";
    cout<<endl;
    for(int i=0;i<calls_for_nt.size();i++)
    {
        cout<<calls_for_nt[i]<<"\t";
        for(string ts:t)
        {
            cout<<parse_table[{calls_for_nt[i],ts}]<<"\t";
        }
        cout<<parse_table[{calls_for_nt[i],"&"}]<<"\t";
        cout<<endl;
    }
    stack.push_back(calls_for_nt[0]+"&");
    cout<<"\n\nENTER STRING TO CHECK : ";
    getline(cin,s);
    cur_str.push_back(s+"&");
    rule_used.push_back(" ");
    check_str();
    cout<<"\n\nSTACK\t\tINPUT\t\tPRODUCTION\n";
    for(int i=0;i<stack.size();i++)
        cout<<stack[i]<<"\t\t"<<cur_str[i]<<"\t\t"<<rule_used[i]<<endl;
    if(stack[stack.size()-1][0]=='&' && cur_str[cur_str.size()-1][0]=='&')
        cout<<"\n\nSTRING IS ACCEPTED !!!";
    else
        cout<<"\nSTRING IS ACCEPTED !!!";
    return 0;
}
```