Lab Assignment 5

U21CS089 | Garvit Shah

1. Write a program to construct LL (1) parse table for the following grammar and check whether the given input can be accepted or not.

Grammar: E --> TE'
E' --> +TE' | ε
T --> FT'
T' --> *FT' | ε
F --> id | (E)
*ε denotes epsilon.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <map>
#include <set>
#include <stack>
#include <string>


using namespace std;


void find_first(vector<pair<char, string> > gram,
                map<char, set<char> > &firsts,
                char non_term)
{
    for (auto it = gram.begin(); it != gram.end(); ++it)
    {
        // Find productions of our non terminal
        if (it->first != non_term)
        {
            continue;
        }


        string rhs = it->second;
        for (auto ch = rhs.begin(); ch != rhs.end(); ++ch)
        {
            //1st rule of finding first and 2nd rule too
            if (!isupper(*ch))
            {
                firsts[non_term].insert(*ch);
                break;
            }
            else
            {

                if (firsts[*ch].empty())
                {
                    find_first(gram, firsts, *ch);
                }
                // If variable doesn't have epsilon, stop loop
                if (firsts[*ch].find('#') == firsts[*ch].end())
                {
                    firsts[non_term].insert(firsts[*ch].begin(),
firsts[*ch].end());
                    break;
```

```
                }


                set<char> firsts_copy(firsts[*ch].begin(), firsts[*ch].end());


                // Remove epsilon from firsts if not the last variable
                //3rd rule of finding first
                if (ch + 1 != rhs.end())
                {
                    firsts_copy.erase('#');
                }


                // Append firsts of that variable
                firsts[non_term].insert(firsts_copy.begin(), firsts_copy.end());
            }
        }
    }
}


void find_follow(vector<pair<char, string> > gram,
                 map<char, set<char> > &follows,
                 map<char, set<char> > firsts,
                 char non_term)
{
    for (auto it = gram.begin(); it != gram.end(); ++it){


        // finished is true when finding follow from this production is complete
        bool finished = true;
        auto ch = it->second.begin();


        // Skip variables till reqd non terminal
        for (; ch != it->second.end(); ++ch){
            if (*ch == non_term)
            {
                finished = false;
                break;
            }
        }
        ++ch;
        for (; ch != it->second.end() && !finished; ++ch){
            // If non terminal, just append to follow
            //2nd rule of follow
            if (!isupper(*ch))
            {
                follows[non_term].insert(*ch);
                finished = true;
                break;
            }


            set<char> firsts_copy(firsts[*ch]);

            if (firsts_copy.find('#') == firsts_copy.end())
            {
                follows[non_term].insert(firsts_copy.begin(), firsts_copy.end());
                finished = true;
                break;
            }
            // Else next char has to be checked after appending firsts to follow
            firsts_copy.erase('#');
            follows[non_term].insert(firsts_copy.begin(), firsts_copy.end());
```

```
        }


        // If end of production, follow same as follow of variable
        // 3rd rule of follow
        if (ch == it->second.end() && !finished)
        {
            // Find follow if it doesn't have
            if (follows[it->first].empty())
            {
                find_follow(gram, follows, firsts, it->first);
            }
            follows[non_term].insert(follows[it->first].begin(), follows[it-
>first].end());
        }
    }
}


int main()
{
    // Parsing the grammar file
    fstream grammar_file;
    grammar_file.open("input.txt");
    if (grammar_file.fail())
    {
        cout << "There is error in opening the given input file. \n";
        return 2;
    }


    cout << "\n\n The grammar parsed is: \n";
     cout << "\n";
    vector<pair<char, string> > gram;
    int count = 0;
    // to store the grammar and show it in terminal
    while (!grammar_file.eof()) // till the end of the file loop will be executed
    {
        char buffer[20];
        grammar_file.getline(buffer, 19);


        char lhs = buffer[0];              // storing non-terminals char
        string rhs = buffer + 3;           // storing production as string
        pair<char, string> prod(lhs, rhs); // making pair of them
        gram.push_back(prod);
        cout << count++ << ".  " << gram.back().first << " -> " <<
gram.back().second << "\n"; // showing it
    }
    cout << "\n";


    // Gather all non terminals
    set<char> n_terminals;
    for (auto i = gram.begin(); i != gram.end(); ++i)
    {
        n_terminals.insert(i->first); // storing non terminals into set
    }

cout<<"*****************************************************************"<<endl;
    cout << "\n The Non-Terminals present in the grammar are: ";

    for (auto i = n_terminals.begin(); i != n_terminals.end(); ++i)
    {
        cout << *i << " ";
    }
```

```
    cout << "\n";

cout<<"\n**********************************************************************"<<endl;
    cout << "\n";
    // Gather all terminals
    set<char> terminals;
    for (auto i = gram.begin(); i != gram.end(); ++i) // loop running in vector
    {
        for (auto ch = i->second.begin(); ch != i->second.end(); ++ch) // loop
running in string
        {
            if (!isupper(*ch))
            {
                terminals.insert(*ch);
            }
        }
    }
    terminals.erase('#');
    terminals.insert('$');
    cout << "The terminals present in the grammar are: ";
    for (auto i = terminals.begin(); i != terminals.end(); ++i)
    {
        cout << *i << " ";
    }
    cout << "\n";

cout<<"\n**********************************************************************"<<endl;
    cout << "\n\n";


    // Start symbol is first non terminal production in grammar
    char start_sym = gram.begin()->first;


    map<char, set<char> > firsts;
    for (auto non_term = n_terminals.begin(); non_term != n_terminals.end(); +
+non_term)
    {
        if (firsts[*non_term].empty())
        {
            find_first(gram, firsts, *non_term);
        }
    }


    cout << "**********FIRST LIST********** \n";
    cout << "\n";
    for (auto it = firsts.begin(); it != firsts.end(); ++it)
    {
        cout << it->first << " : {";
        for (auto firsts_it = it->second.begin(); firsts_it != it->second.end();
++firsts_it)
        {
            if (firsts_it != prev(it->second.end()))
                cout << *firsts_it << ",";
            else
                cout << *firsts_it;
        }
        cout << "}\n";
    }

cout<<"\n**********************************************************************"<<endl;
    cout << "\n\n";


    map<char, set<char> > follows;
```

```
    // Find follow of start variable first
    char start_var = gram.begin()->first;
    follows[start_var].insert('$'); // first rule of follow
    find_follow(gram, follows, firsts, start_var);
    // Find follows for rest of variables
    for (auto it = n_terminals.begin(); it != n_terminals.end(); ++it)
    {
        if (follows[*it].empty())
        {
            find_follow(gram, follows, firsts, *it);
        }
    }


    cout << "**********FOLLOW LIST********** \n";
    cout << "\n";
    for (auto it = follows.begin(); it != follows.end(); ++it)
    {
        cout << it->first << " : {";
        for (auto follows_it = it->second.begin(); follows_it != it-
>second.end(); ++follows_it)
        {
            if (follows_it != prev(it->second.end()))
                cout << *follows_it << ",";
            else
                cout << *follows_it;
        }
        cout << "}\n";
    }

cout<<"\n**********************************************************************"<<endl;
    cout << "\n\n";


    int parse_table[n_terminals.size()][terminals.size()];
    fill(&parse_table[0][0], &parse_table[0][0] + sizeof(parse_table) /
sizeof(parse_table[0][0]), -1);
    for (auto prod = gram.begin(); prod != gram.end(); ++prod)
    {
        string rhs = prod->second;


        set<char> next_list;
        bool finished = false;
        for (auto ch = rhs.begin(); ch != rhs.end(); ++ch)
        {
            if (!isupper(*ch))
            {
                if (*ch != '#')
                {
                    next_list.insert(*ch);
                    finished = true;
                    break;
                }
                continue;
            }


            set<char> firsts_copy(firsts[*ch].begin(), firsts[*ch].end());
            if (firsts_copy.find('#') == firsts_copy.end())
            {
                next_list.insert(firsts_copy.begin(), firsts_copy.end());
                finished = true;
                break;
            }
            firsts_copy.erase('#');
```

```
            next_list.insert(firsts_copy.begin(), firsts_copy.end());
        }
        // If the whole rhs can be skipped through epsilon or reaching the end
        // Add follow to next list
        if (!finished)
        {
            next_list.insert(follows[prod->first].begin(), follows[prod-
>first].end());
        }


        for (auto ch = next_list.begin(); ch != next_list.end(); ++ch)
        {
            int row = distance(n_terminals.begin(), n_terminals.find(prod-
>first));
            int col = distance(terminals.begin(), terminals.find(*ch));
            int prod_num = distance(gram.begin(), prod);
            if (parse_table[row][col] != -1)
            {
                cout << "COLLISION AT [" << row << "][" << col << "] FOR
PRODUCTION " << prod_num << "\n";
                continue;
            }
            parse_table[row][col] = prod_num;
        }
    }
    cout << "**********PARSING TABLE********** \n";
    cout << "\n";
    cout << "    ";
    for (auto i = terminals.begin(); i != terminals.end(); ++i)
    {
        cout << *i << " ";
    }
    cout << "\n";
    for (auto row = n_terminals.begin(); row != n_terminals.end(); ++row)
    {
        cout << *row << "  ";
        for (int col = 0; col < terminals.size(); ++col)
        {
            int row_num = distance(n_terminals.begin(), row);
            if (parse_table[row_num][col] == -1)
            {
                cout << "- ";
                continue;
            }
            cout << parse_table[row_num][col] << " ";
        }
        cout << "\n";
    }
    cout << "\n";
    string s;

cout<<"\n*****************************************************************"<<endl;
    cout<<"\nEnter the string to be parsed :";
    cin>>s;
    string input_string(s);
    input_string.push_back('$');
    stack<char> st;
    st.push('$');
    st.push(start_sym);

    cout << "\n";


    // Check if input string is valid
    for (auto ch = input_string.begin(); ch != input_string.end(); ++ch)
```

```
    {
        if (terminals.find(*ch) == terminals.end())
        {
            cout << "The given input string is invalid. \n";
            return 2;
        }
    }


    // cout<<"Processing input string\n";
    bool accepted = true;
    while (!st.empty() && !input_string.empty())
    {
        // If stack top same as input string char remove it


        if (input_string[0] == st.top())
        {
            st.pop();
            input_string.erase(0, 1);
        }
        else if (!isupper(st.top()))
        {
            cout << "There is no matching terminal found. \n";
            accepted = false;
            break;
        }
        else
        {
            char stack_top = st.top();
            int row = distance(n_terminals.begin(), n_terminals.find(stack_top));
            int col = distance(terminals.begin(),
terminals.find(input_string[0]));
            int prod_num = parse_table[row][col];


            if (prod_num == -1)
            {
                cout << "There is no such production in the table\n";
                accepted = false;
                break;
            }


            st.pop();
            string rhs = gram[prod_num].second;
            if (rhs[0] == '#')
            {
                continue;
            }
            for (auto ch = rhs.rbegin(); ch != rhs.rend(); ++ch)
            {
                st.push(*ch);
            }
        }
    }
    if (accepted){
        cout << "The input string is accepted  by the grammar.\n\n";
    }
    else{
        cout << "The input string is rejected  by the grammar.\n\n";
    }
    cout <<"\n";
    return 0;
}
```

```
» ~/D/C/V/S/L/Assign2 cd "/Users/garvitshah/Desktop/College/VI/SS/Lab/Assign5/" && g++ --std=c++17 LL1.cpp


 The grammar parsed is:

0.  A -> CB
1.  B -> +CB
2.  B -> #
3.  C -> ED
4.  D -> *ED
5.  D -> #
6.  E -> i
7.  E -> (A)

****************************************************************

 The Non-Terminals present in the grammar are: A B C D E

****************************************************************

 $ ( ) * + i  present in the grammar are:

****************************************************************


**********FIRST LIST**********

A : {(,i}
B : {#,+}
C : {(,i}
D : {#,*}
E : {(,i}

****************************************************************


**********FOLLOW LIST**********

A : {$,)}
} : {
,+} {
} : {
,*} {

****************************************************************


**********PARSING TABLE**********

 $ ( ) * + i
A  - - 0 - - - 0
B  2 - - - - 1 -
C  - - 3 - - - 3
D  5 - - - 4 - -
E  - - 7 - - - 6


****************************************************************

Enter the string to be parsed :i

There is no matching terminal found.
```