**Compiler Design Lab**

**Name –** N Aditya Sai

**Register No –** RA1911030010075

**Experiment –** 6

**Aim:** A program to implement Predictive Parser Table

**Algorithm:**

**FIRST(A):**

Step 1:  First check for left recursion in the grammar, if there is left recursion in the grammar remove that and go to step 2.

Step 2: Calculate First() and Follow() for all non-terminals.

 First(): If there is a variable, and from that variable, if we try to drive all the strings then the beginning Terminal Symbol is called the First.

Follow(): What is the Terminal Symbol which follows a variable in the process of derivation.

Step 3: For each production A –> α. (A tends to alpha)

1. Find First(α) and for each terminal in First(α), make entry A –> α in the table.

2. If First(α) contains ε (epsilon) as terminal than, find the Follow(A) and for each terminal in Follow(A), make entry A –> α in the table.

3. If the First(α) contains ε and Follow(A) contains $ as terminal, then make entry A –> α in the table

**Code:**

#include<iostream>

#include<string>

#include<deque>

using namespace std;

int n,n1,n2;

int getPosition(string arr[], string q, int size)

{

for(int i=0;i<size;i++)

{

if(q == arr[i])

return i;

}

return -1;

}

int main()

{

string prods[10],first[10],follow[10],nonterms[10],terms[10];

string pp\_table[20][20] = {};

cout<<"Enter the number of productions : ";

cin>>n;

cin.ignore();

cout<<"Enter the productions"<<endl;

for(int i=0;i<n;i++)

{

getline(cin,prods[i]);

cout<<"Enter first for "<<prods[i].substr(3)<<" : ";

getline(cin,first[i]);

}

cout<<"Enter the number of Terminals : ";

cin>>n2;

cin.ignore();

cout<<"Enter the Terminals"<<endl;

for(int i=0;i<n2;i++)

{

cin>>terms[i];

}

terms[n2] = "$";

n2++;

cout<<"Enter the number of Non-Terminals : ";

cin>>n1;

cin.ignore();

for(int i=0;i<n1;i++)

{

cout<<"Enter Non-Terminal : ";

getline(cin,nonterms[i]);

cout<<"Enter follow of "<<nonterms[i]<<" : ";

getline(cin,follow[i]);

}

cout<<endl;

cout<<"Grammar"<<endl;

for(int i=0;i<n;i++)

{

cout<<prods[i]<<endl;

}

for(int j=0;j<n;j++)

{

int row = getPosition(nonterms,prods[j].substr(0,1),n1);

if(prods[j].at(3)!='#')

{

for(int i=0;i<first[j].length();i++)

{

int col = getPosition(terms,first[j].substr(i,1),n2);

pp\_table[row][col] = prods[j];

}

}

else

{

for(int i=0;i<follow[row].length();i++)

{

int col = getPosition(terms,follow[row].substr(i,1),n2);

pp\_table[row][col] = prods[j];

}

}

}

//Display Table

for(int j=0;j<n2;j++)

cout<<"\t"<<terms[j];

cout<<endl;

for(int i=0;i<n1;i++)

{

cout<<nonterms[i]<<"\t";

//Display Table

for(int j=0;j<n2;j++)

{

cout<<pp\_table[i][j]<<"\t";

}

cout<<endl;

}

//Parsing String

char c;

do{

string ip;

deque<string> pp\_stack;

pp\_stack.push\_front("$");

pp\_stack.push\_front(prods[0].substr(0,1));

cout<<"Enter the string to be parsed : ";

getline(cin,ip);

ip.push\_back('$');

cout<<"Stack\tInput\tAction"<<endl;

while(true)

{

for(int i=0;i<pp\_stack.size();i++)

cout<<pp\_stack[i];

cout<<"\t"<<ip<<"\t";

int row1 = getPosition(nonterms,pp\_stack.front(),n1);

int row2 = getPosition(terms,pp\_stack.front(),n2);

int column = getPosition(terms,ip.substr(0,1),n2);

if(row1 != -1 && column != -1)

{

string p = pp\_table[row1][column];

if(p.empty())

{

cout<<endl<<"String cannot be Parsed."<<endl;

break;

}

pp\_stack.pop\_front();

if(p[3] != '#')

{

for(int x=p.size()-1;x>2;x--)

{

pp\_stack.push\_front(p.substr(x,1));

}

}

cout<<p;

}

else

{

if(ip.substr(0,1) == pp\_stack.front())

{

if(pp\_stack.front() == "$")

{

cout<<endl<<"String Parsed."<<endl;

break;

}

cout<<"Match "<<ip[0];

pp\_stack.pop\_front();

ip = ip.substr(1);

}

else

{

cout<<endl<<"String cannot be Parsed."<<endl;

break;

}

}

cout<<endl;

}

cout<<"Continue?(Y/N) ";

cin>>c;

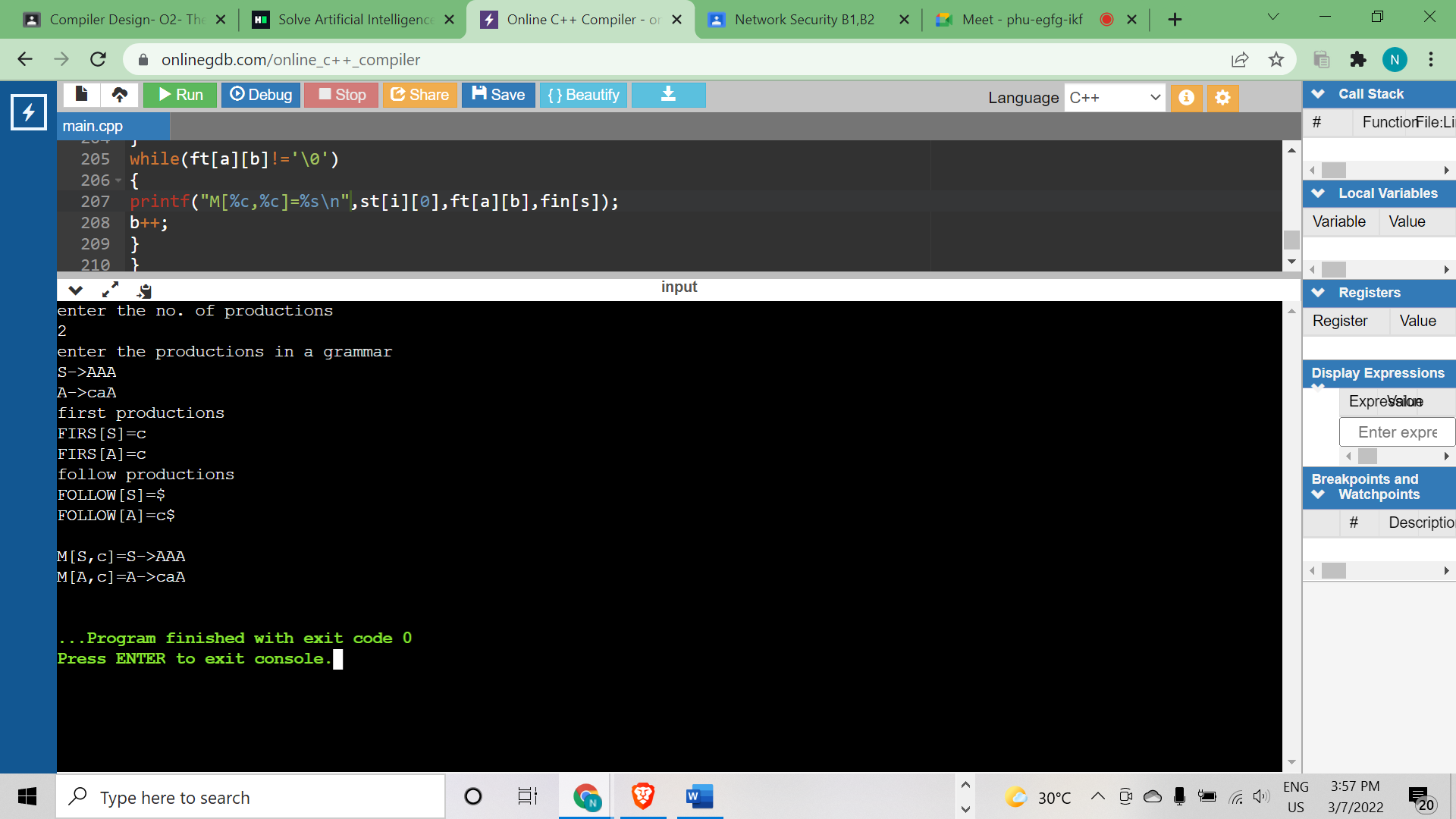
cin.ignore();

}while(c=='y' || c=='Y');

return 0;

}

**Screenshots and Output:**



**Result**: Predictive Parser Table has been implemented successfully using C++.