**COMPILER DESIGN PRACTICAL LAB-4**

**Elimination of Left Recursion and Left Factoring**

SANDRA MARIA TONY RA1911026010045 K1

**ELIMINATION OF LEFT RECURSION**

**AIM:** A program for Elimination of Left Recursion.

**THEORY:**

* For a given grammar if the leftmost variable of its RHS is similar to its variable to its LHS then it has left recursion.
* So, if a grammar having a left recursion then it is called Left Recursive Grammar.

Therefore a grammar is said to be left recursive if the leftmost symbol on the right side is equivalent to the nonterminal on the left side. For Ex: exp → exp + term.

**ALGORITHM:**

* Start the program.
* Initialize the arrays for taking input from the user.
* Prompt the user to input the no. of non-terminals having left recursion and no. of productions for these non-terminals.
* Prompt the user to input the production for non-terminals.
* Eliminate left recursion using the following rules:- A->Aα1| Aα2 | . . . . . |Aαm A->β1| β2| . . . . .| βn Then replace it by A-> βi A’ i=1,2,3,…..m A’-> αj A’ j=1,2,3,…..n A’-> Ɛ
* After eliminating the left recursion by applying these rules, display the productions without left recursion.
* Exit

**CODE:**

#include <iostream>

#include <string>

using namespace std;

int main()

{

int n, j, l, i, k;

int length[10] = {};

string d, a, b, flag;

char c;

cout<<"Enter Parent Non-Terminal: ";

cin >> c;

d.push\_back(c);

a += d + "\'->";

d += "->";

b += d;

cout<<"Enter productions: ";

cin >> n;

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

{

cout<<"Enter Production ";

cout<<i + 1<<" :";

cin >> flag;

length[i] = flag.size();

d += flag;

if (i != n - 1)

{

d += "|";

}

}

cout<<"The Production Rule is: ";

cout<<d<<endl;

for (i = 0, k = 3; i < n; i++)

{

if (d[0] != d[k])

{

cout<<"Production: "<< i + 1;

cout<<" does not have left recursion.";

cout<<endl;

if (d[k] == '#')

{

b.push\_back(d[0]);

b += "\'";

}

else

{

for (j = k; j < k + length[i]; j++)

{

b.push\_back(d[j]);

}

k = j + 1;

b.push\_back(d[0]);

b += "\'|";

}

}

else

{

cout<<"Production: "<< i + 1 ;

cout<< " has left recursion";

cout<< endl;

if (d[k] != '#')

{

for (l = k + 1; l < k + length[i]; l++)

{

a.push\_back(d[l]);

}

k = l + 1;

a.push\_back(d[0]);

a += "\'|";

}

}

}

a += "#";

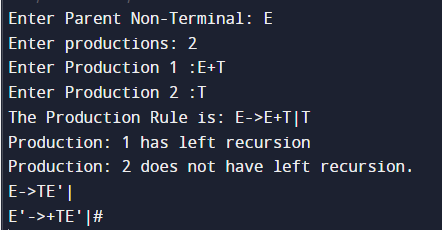
cout << b << endl;

cout << a << endl;

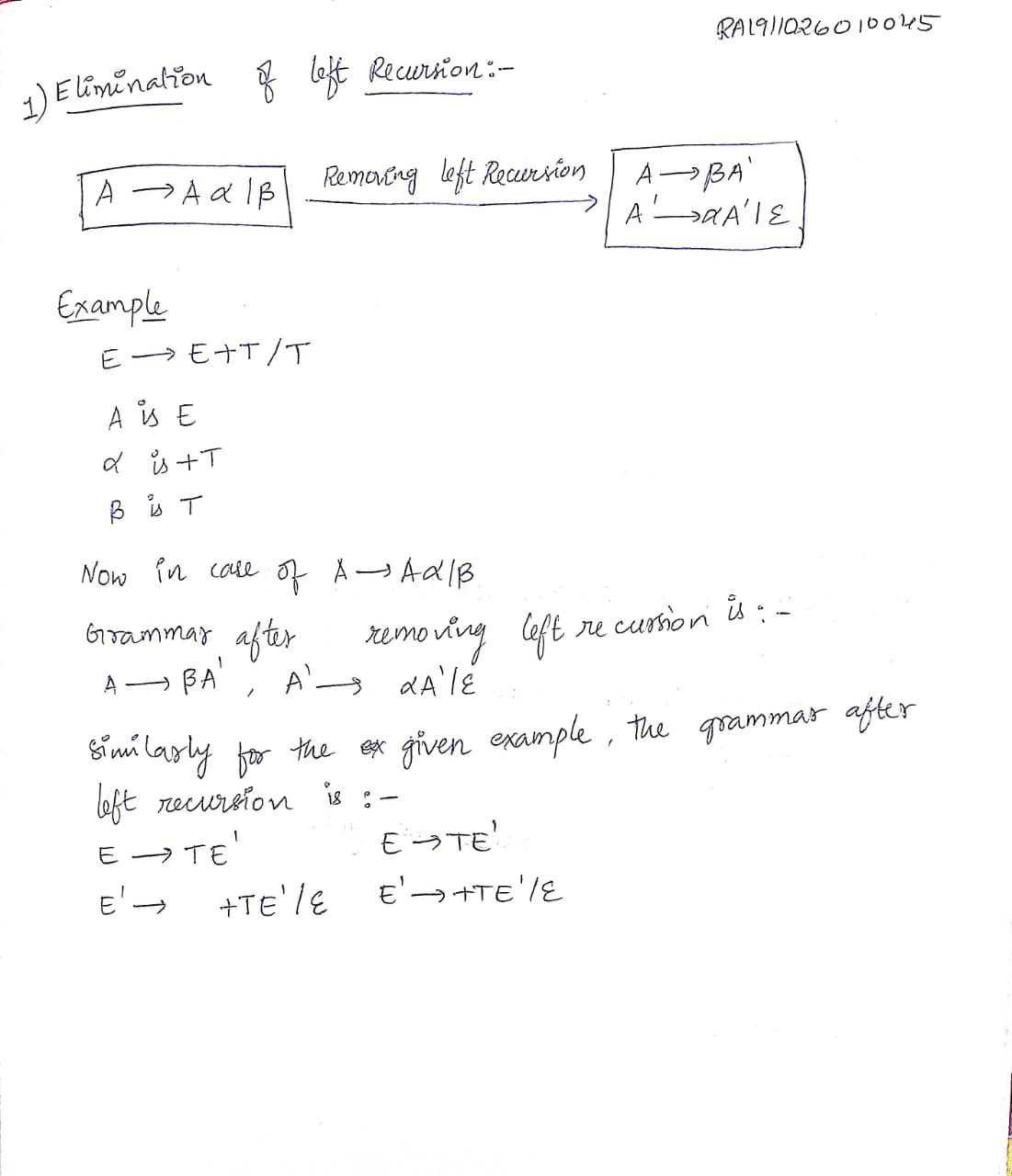
return 0;

}

**OUTPUT:**

****

**MANUAL CALCULATION:**



**RESULT:**

A program for Elimination of Left Recursion was executed successfully.

**LEFT FACTORING**

**AIM:** A program for implementation Of Left Factoring

**THEORY:**

**In the left factoring**

* For the common prefixes, we make only 1 production.
* So, here the common prefix can be a terminal or a non-terminal or it can be a combination of both.
* With the help of new productions, the rest derivation is added.

The result that is obtained after this process of left factoring is known as Left Factored Grammar.

**ALGORITHM:**

1. Start

2. Ask the user to enter the set of productions

3. Check for common symbols in the given set of productions by comparing with: A->aB1|aB2

4. If found, replace the particular productions with: A->aA’ A’->B1 | B2|ɛ

5. Display the output

6. Exit

**CODE:**

#include <iostream>

#include <string>

using namespace std;

int main()

{

int n,j,l,i,m;

int len[10] = {};

string a, b1, b2, flag;

char c;

cout << "Enter the Parent Non-Terminal : ";

cin >> c;

a.push\_back(c);

b1 += a + "\'->";

b2 += a + "\'\'->";;

a += "->";

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

cin >> n;

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

{

cout << "Enter the Production " << i + 1 << " : ";

cin >> flag;

len[i] = flag.size();

a += flag;

if (i != n - 1)

{

a += "|";

}

}

cout << "The Production Rule is : " << a << endl;

char x = a[3];

for (i = 0, m = 3; i < n; i++)

{

if (x != a[m])

{

while (a[m++] != '|');

}

else

{

if (a[m + 1] != '|')

{

b1 += "|" + a.substr(m + 1, len[i] - 1);

a.erase(m - 1, len[i] + 1);

}

else

{

b1 += "#";

a.insert(m + 1, 1, a[0]);

a.insert(m + 2, 1, '\'');

m += 4;

}

}

}

char y = b1[6];

for (i = 0, m = 6; i < n - 1; i++)

{

if (y == b1[m])

{

if (b1[m + 1] != '|')

{

flag.clear();

for (int s = m + 1; s < b1.length(); s++)

{

flag.push\_back(b1[s]);

}

b2 += "|" + flag;

b1.erase(m - 1, flag.length() + 2);

}

else

{

b1.insert(m + 1, 1, b1[0]);

b1.insert(m + 2, 2, '\'');

b2 += "#";

m += 5;

}

}

}

b2.erase(b2.size() - 1);

cout << "After Left Factoring : " << endl;

cout << a << endl;

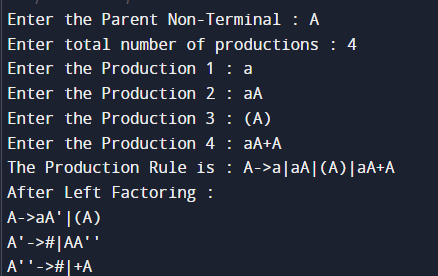
cout << b1 << endl;

cout << b2 << endl;

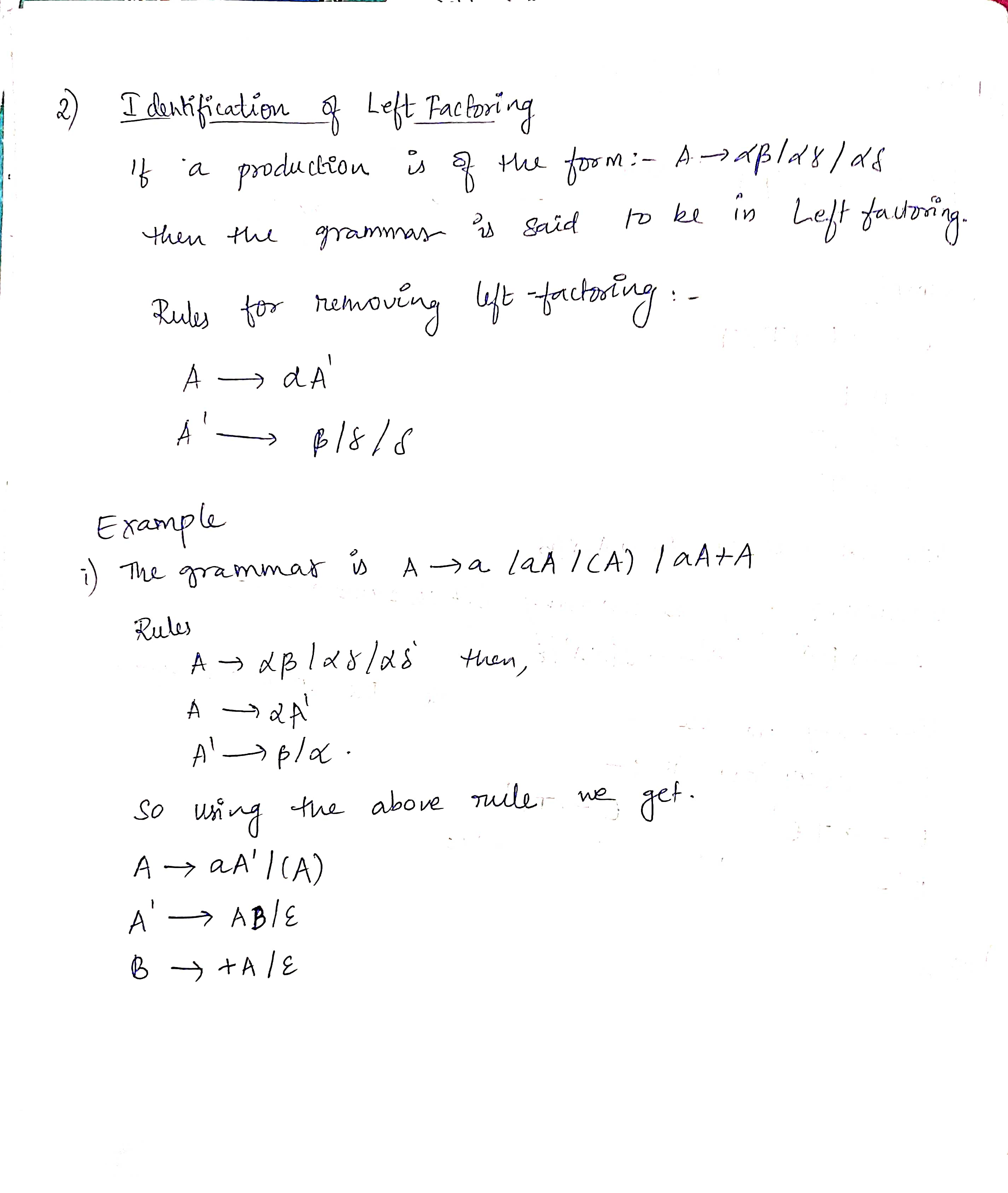
return 0;

}

**OUTPUT:**



**MANUAL CALCULATION:**

**RESULT:**

A program for implementation of Left Factoring was compiled and executed successfully.