**Compiler Design Lab**

**Name –** Sai Dhatri Byrisetti

**Register No –** RA1911030010087

**Topic** - Removal of Left Recursion and Left Factors

**Experiment –** 4A

**Aim:** To study the removal of left recursion and write a program in order to remove left recursions from a given set of productions.

**Algorithm**:

* Start
* Prompt the user to enter the number of production lines.
* For each line of production, call a function which removes left recursion from each production line.
* In the function, ask the user to enter the grammar line.
* 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.
* Stop

**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 += "#";

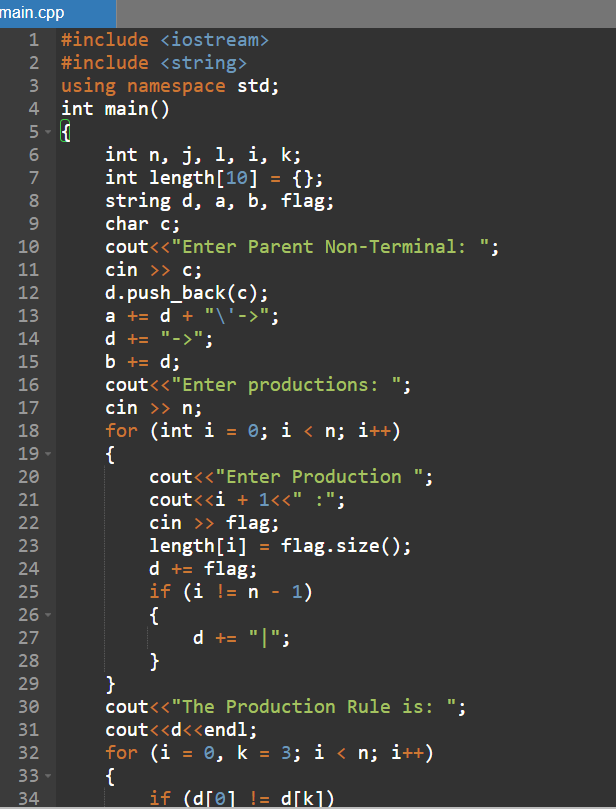
cout << b << endl;

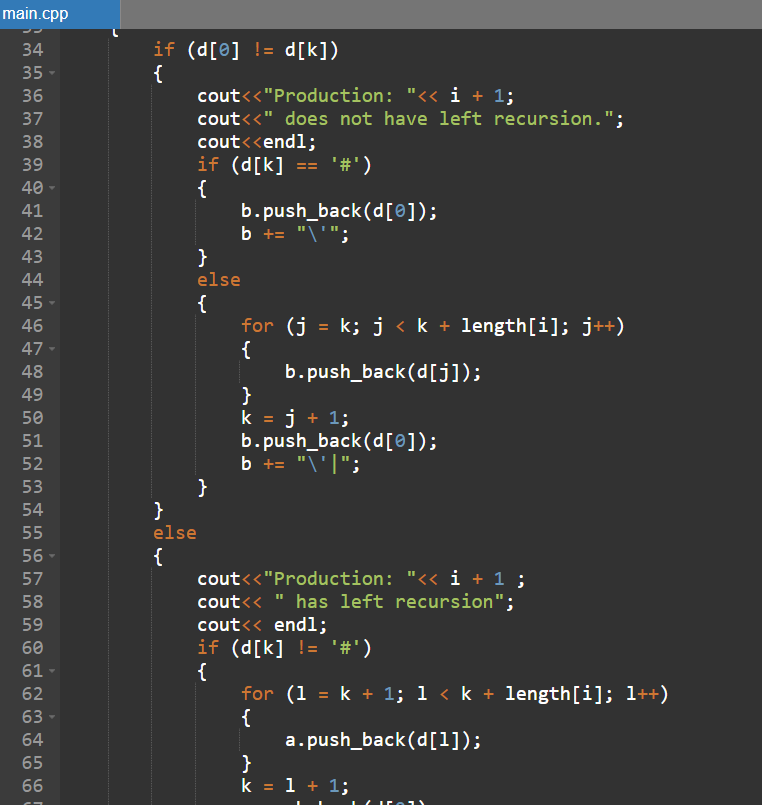
cout << a << endl;

return 0;

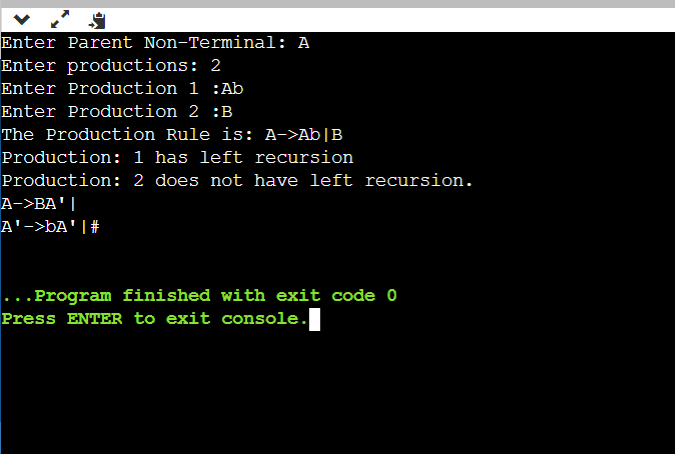
}

**Screenshots:**

****

****

**Output:**



**Experiment –** 4B

**Aim:** To study the removal of left factors and write a program in order to remove left factors from a given set of productions.

**Algorithm:**

* Start
* Ask the user to enter the set of productions.
* Check for common symbols in the given set of productions by comparing with:

A-> aB1 | aB2

* If found, replace the particular productions with:

A -> aA’

A’ -> B1 | B2 | Ɛ

* Display the output.
* Exit

**Code:**

#include<iostream>

#include<string>

using namespace std;

int main()

{ string ip,op1,op2,temp;

int sizes[10] = {};

char c;

int n,j,l;

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

cin>>c;

ip.push\_back(c);

op1 += ip + "\'->";

op2 += ip + "\'\'->";;

ip += "->";

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

cin>>n;

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

{

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

cin>>temp;

sizes[i] = temp.size();

ip+=temp;

if(i!=n-1)

ip += "|";

}

cout<<"Production Rule : "<<ip<<endl;

char x = ip[3];

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

{

if(x == ip[k])

{

if(ip[k+1] == '|')

{

op1 += "#";

ip.insert(k+1,1,ip[0]);

ip.insert(k+2,1,'\'');

k+=4;

}

else

{

op1 += "|" + ip.substr(k+1,sizes[i]-1);

ip.erase(k-1,sizes[i]+1);

}

}

else

{

while(ip[k++]!='|');

}

}

char y = op1[6];

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

{

if(y == op1[k])

{

if(op1[k+1] == '|')

{

op2 += "#";

op1.insert(k+1,1,op1[0]);

op1.insert(k+2,2,'\'');

k+=5;

}

else

{

temp.clear();

for(int s=k+1;s<op1.length();s++)

temp.push\_back(op1[s]);

op2 += "|" + temp;

op1.erase(k-1,temp.length()+2);

} }}

op2.erase(op2.size()-1);

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

cout<<ip<<endl;

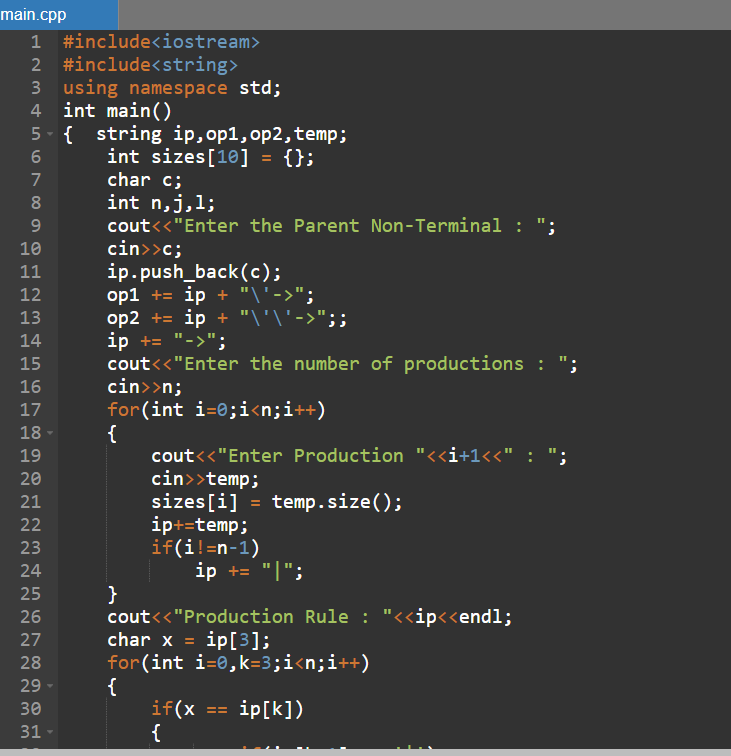
cout<<op1<<endl;

cout<<op2<<endl;

return 0;

}

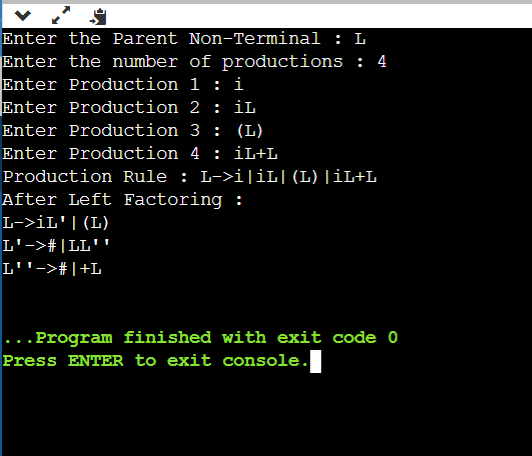
**Screenshots**:







**Output:**



**Result:** Removal of left recursion and left factors was studied and implemented in C++ successfully.