Exp-4a Elimination of Left Recursion

Name: - Sai Mohit Ambekar

Reg No: - RA1911031010137

Class: - CSE-IT (L2 Section)

AIM: A program for Elimination of Left Recursion.

ALGORITHM:

1. Start the program.
2. Initialize the arrays for taking input from the user.
3. Prompt the user to input the no. of non-terminals having left recursion and no. of productions for these non-terminals.
4. Prompt the user to input the production for non-terminals.
5. 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
6. After eliminating the left recursion by applying these rules, display the productions without left recursion.
7. Stop.

PROGRAM:

```
#include <iostream>
```

```cpp
#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 + "\'->";
    ip += "->";
    op2 += 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;
    for (int i = 0, k = 3; i < n; i++)
    {
        if (ip[0] == ip[k])
        {
            cout << "Production " << i + 1 << " has left recursion." << endl;
            if (ip[k] != '#')
            {
                for (l = k + 1; l < k + sizes[i]; l++)
                    op1.push_back(ip[l]);
```

```
            k = l + 1;
            op1.push_back(ip[0]);
            op1 += "\'|";
        }
    }
    else
    {
        cout << "Production " << i + 1 << " does not have left recursion." << endl;
        if (ip[k] != '#')
        {
            for (j = k; j < k + sizes[i]; j++)
                op2.push_back(ip[j]);
            k = j + 1;
            op2.push_back(ip[0]);
            op2 += "\'|";
        }
        else
        {
            op2.push_back(ip[0]);
            op2 += "\'";
        }
    }
  }
  op1 += "#";
  cout << op2 << endl;
  cout << op1 << endl;
  return 0;
}
```

OUTPUT:

RESULT:

A program for Elimination of Left Recursion was run successfully.

Exp-4b Elimination of Left Factoring

AIM: A program for implementation Of Left Factoring

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|ε

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:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                   Code - EXP 4

cd "/Users/saimohitambekar/Documents/Sai Work/Class/Compiler Design Lab/EXP 4/" && g++ left_factoring.cpp -o left_factoring && "/Users/saimohitambekar
saimohitambekar@Sais-Air Compiler Design Lab % cd "/Users/saimohitambekar/Documents/Sai Work/Class/Compiler Design Lab/EXP 4/" && g++ left_factoring.cpp -o
left_factoring && "/Users/saimohitambekar/Documents/Sai Work/Class/Compiler Design Lab/EXP 4/"left_factoring
Enter the Parent Non-Terminal: M
Enter total number of productions: 4
Enter the Production 1 : i
Enter the Production 2 : iM
Enter the Production 3 : (M)
Enter the Production 4 : iM+M
The Production Rule is: M->i|iM|(M)|iM+M
After Left Factoring:
M->iM'|(M)
M'->#|MM''
M''->#|+M
saimohitambekar@Sais-Air EXP 4 %
```

RESULT:

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