Elimination of Left Recursion and Left Factoring
Compiler Design – Lab 6

Name: Ojas Patil
Reg No: 21BAI1106

AIM

A. Write a C / C++ / Java program to Eliminate Left Recursion from any given Production.
B. Write a C / C++ / Java program to Eliminate Left Factoring from any given Production.

PART A

Algorithm

1. **Input**: Prompt the user to enter the number of productions (**num**).
2. **Input Grammar**: Receive the grammar productions (**production[i]**) from the user.
3. **Iterate Through Productions**: For each production, do the following:
4. **Check Left Recursion**: Determine if the current production exhibits left recursion.
5. **Process Left Recursion**: If left recursive, eliminate left recursion and display the modified grammar.
6. **Continue or Skip**: Decide whether to proceed with the next production or skip if no left recursion is detected.
7. **Repeat**: Repeat steps 3-6 until all productions have been examined.

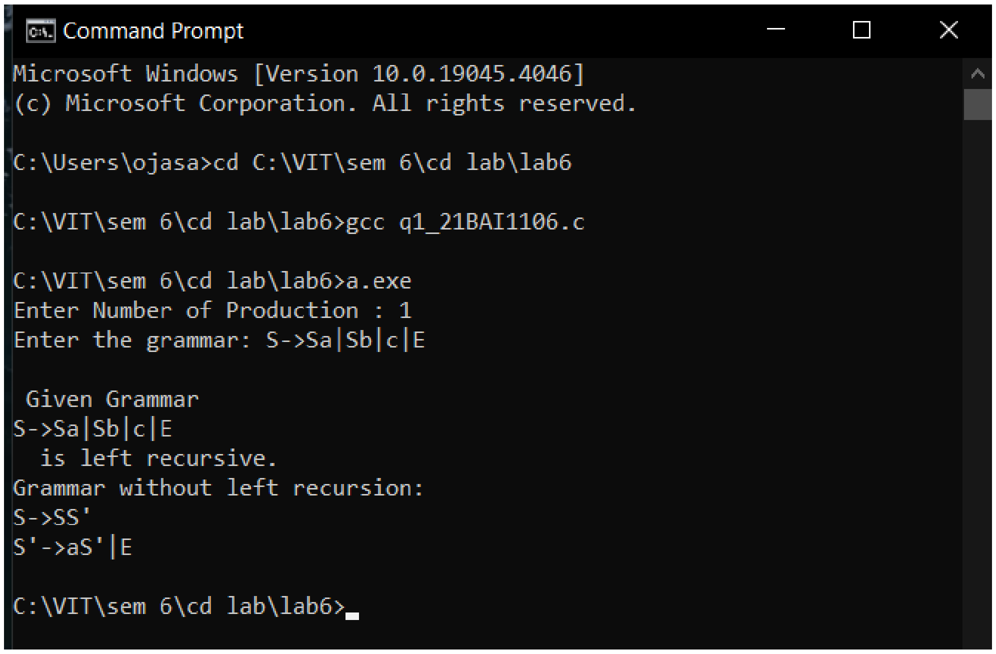
Explanation

The program begins by prompting the user to input the number of productions and the grammar itself. It then iterates through each production, identifying and handling left recursion where applicable. Left recursion removal involves constructing new productions to replace the left-recursive ones. Finally, the algorithm repeats this process until all productions have been processed, ensuring the resulting grammar is free from left recursion.

Sample Input

```
S->Sa| Sb| c | epsilon
```

Output Screenshot



```
Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ojasa>cd C:\VIT\sem 6\cd lab\lab6

C:\VIT\sem 6\cd lab\lab6>gcc q1_21BAI1106.c

C:\VIT\sem 6\cd lab\lab6>a.exe
Enter Number of Production : 1
Enter the grammar: S->Sa|Sb|c|E

 Given Grammar
S->Sa|Sb|c|E
  is left recursive.
Grammar without left recursion:
S->SS'
S'->aS'|E

C:\VIT\sem 6\cd lab\lab6>
```

Source Code

```c
#include <stdio.h>
#include <string.h>
#define SIZE 10

int main () {
    char non_terminal;
    char beta,alpha;
    int num;
    char production[10][SIZE];
    int index=3; /* starting of the string following "->" */
     printf("Enter Number of Production : ");
     scanf("%d",&num);
     printf("Enter the grammar: ");
     for(int i=0;i<num;i++){
          scanf("%s",production[i]);
    }
    for(int i=0;i<num;i++){
        printf("\n Given Grammar \n%s\n ", production[i]);
        non_terminal=production[i][0];
        if(non_terminal==production[i][index]) {
            alpha=production[i][index+1];
            printf(" is left recursive.\n");
            while(production[i][index]!=0 && production[i][index]!='|')
                index++;
            if(production[i][index]!=0) {
                beta=production[i][index+1];
                printf("Grammar without left recursion:\n");
                printf("%c->%c%c\'",non_terminal,beta,non_terminal);
                printf("\n%c\'-
>%c%c\'|E\n",non_terminal,alpha,non_terminal);
            }
            else
                printf(" can't be reduced\n");
        }
        else
            printf(" is not left recursive.\n");
        index=3;
    }
}
```

Part B

Algorithm

1. **Input Nonterminals and Productions**: Prompt the user to input the number of nonterminals and their respective productions.
2. **Identify Common Prefixes**: Iterate through each nonterminal's productions and identify any common prefixes.
3. **Adjust Productions**: For each nonterminal with common prefixes, modify its productions to eliminate the common prefix.
4. **Update Productions**: Generate new productions for modified nonterminals and update the production counts accordingly.
5. **Display Resulting Productions**: Print the resulting modified productions, ensuring each nonterminal and its productions are appropriately formatted.

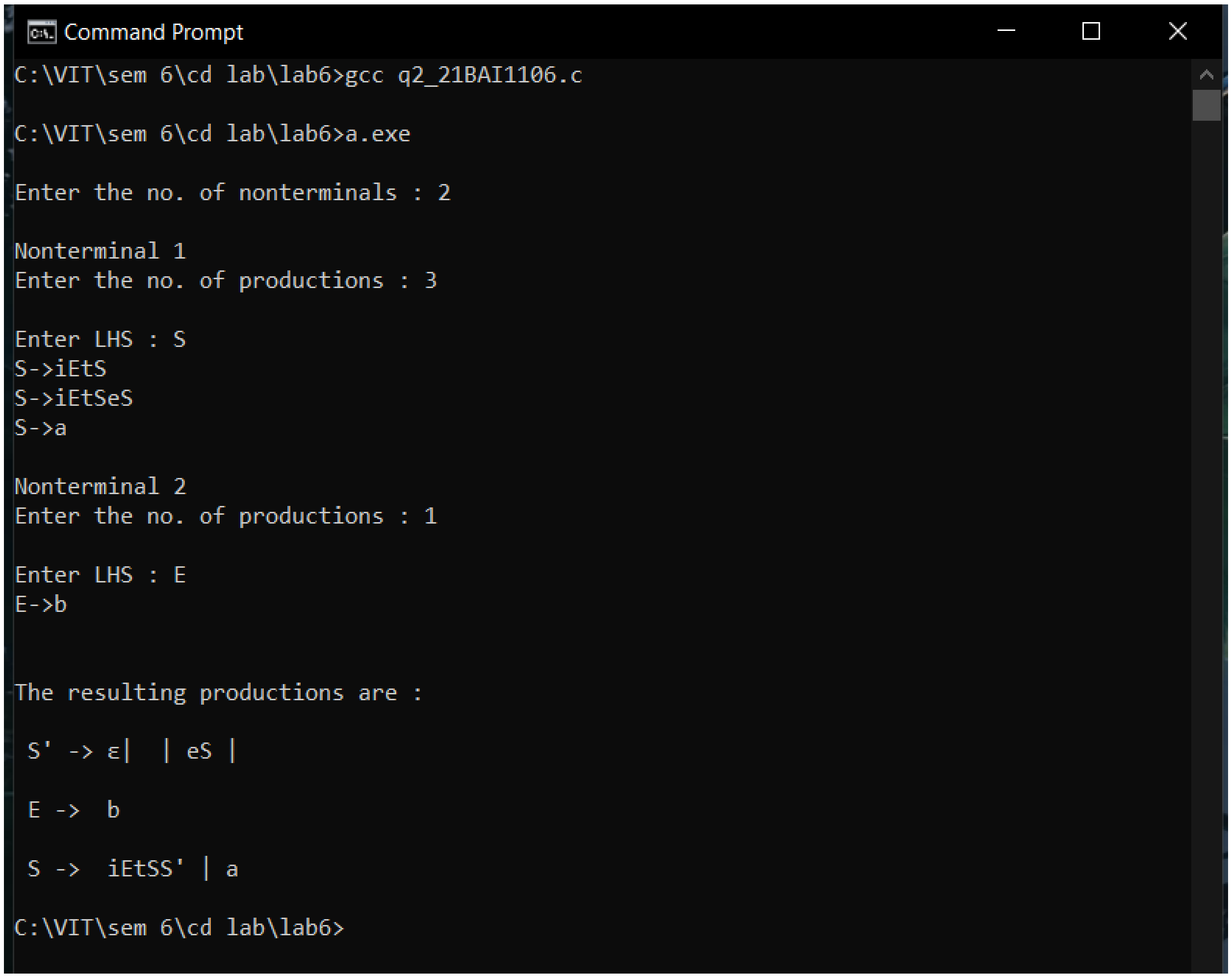
Explanation

The algorithm iterates through each nonterminal's productions, identifying and removing common prefixes. It then generates new productions for modified nonterminals and updates production counts accordingly. Finally, it displays the resulting modified productions, ensuring clear formatting for each nonterminal and its productions. This process simplifies the grammar by eliminating common prefixes, aiding in parsing and analysis tasks.

Sample Input

```
S → iEtS / iEtSeS / a
E → b
```

Output Screenshot



```
C:\VIT\sem 6\cd lab\lab6>gcc q2_21BAI1106.c

C:\VIT\sem 6\cd lab\lab6>a.exe

Enter the no. of nonterminals : 2

Nonterminal 1
Enter the no. of productions : 3

Enter LHS : S
S->iEtS
S->iEtSeS
S->a

Nonterminal 2
Enter the no. of productions : 1

Enter LHS : E
E->b


The resulting productions are :

 S' -> ε|  | eS |

 E ->  b

 S ->  iEtSS' | a

C:\VIT\sem 6\cd lab\lab6>
```

Source Code

```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main()
{
    char ch,lhs[20][20],rhs[20][20][20],temp[20],temp1[20];
    int n,n1,count[20],x,y,i,j,k,c[20];
    printf("\nEnter the no. of nonterminals : ");
    scanf("%d",&n);
    n1=n;
    for(i=0;i<n;i++)
    {
        printf("\nNonterminal %d \nEnter the no. of productions : ",i+1);
        scanf("%d",&c[i]);
        printf("\nEnter LHS : ");
        scanf("%s",lhs[i]);
        for(j=0;j<c[i];j++)
        {
            printf("%s->",lhs[i]);
            scanf("%s",rhs[i][j]);
        }
    }
    for(i=0;i<n;i++)
    {
        count[i]=1;
        while(memcmp(rhs[i][0],rhs[i][1],count[i])==0)
            count[i]++;
    }
    for(i=0;i<n;i++)
    {
        count[i]--;
        if(count[i]>0)
        {
            strcpy(lhs[n1],lhs[i]);
            strcat(lhs[i],"'");
            for(k=0;k<count[i];k++)
                temp1[k] = rhs[i][0][k];
            temp1[k++] = '\0';
            for(j=0;j<c[i];j++)
            {
                for(k=count[i],x=0;k<strlen(rhs[i][j]);x++,k++)
                    temp[x] = rhs[i][j][k];
```

```c
                temp[x++] = '\0';
                if(strlen(rhs[i][j])==1)
                    strcpy(rhs[n1][1],rhs[i][j]);
                strcpy(rhs[i][j],temp);
            }
            c[n1]=2;
            strcpy(rhs[n1][0],temp1);
            strcat(rhs[n1][0],lhs[n1]);
            strcat(rhs[n1][0],"'");
            n1++;
        }
    }
    printf("\n\nThe resulting productions are : \n");
    for(i=0;i<n1;i++)
    {
        if(i==0)
            printf("\n %s -> %c|",lhs[i],(char)238);
        else
            printf("\n %s -> ",lhs[i]);
        for(j=0;j<c[i];j++)
        {
            printf(" %s ",rhs[i][j]);
            if((j+1)!=c[i])
                printf("|");
        }
        printf("\b\b\b\n");
    }
}
```

Result

Thus, we have written two C programs, one to perform Elimination of Left Recursion and one to perform Elimination of Left Factoring. Both these codes outputs have been tested and verified.