RA1911026010033

Compiler Design

Week - 4

Topic : Removing Left Recursion And Left Factoring

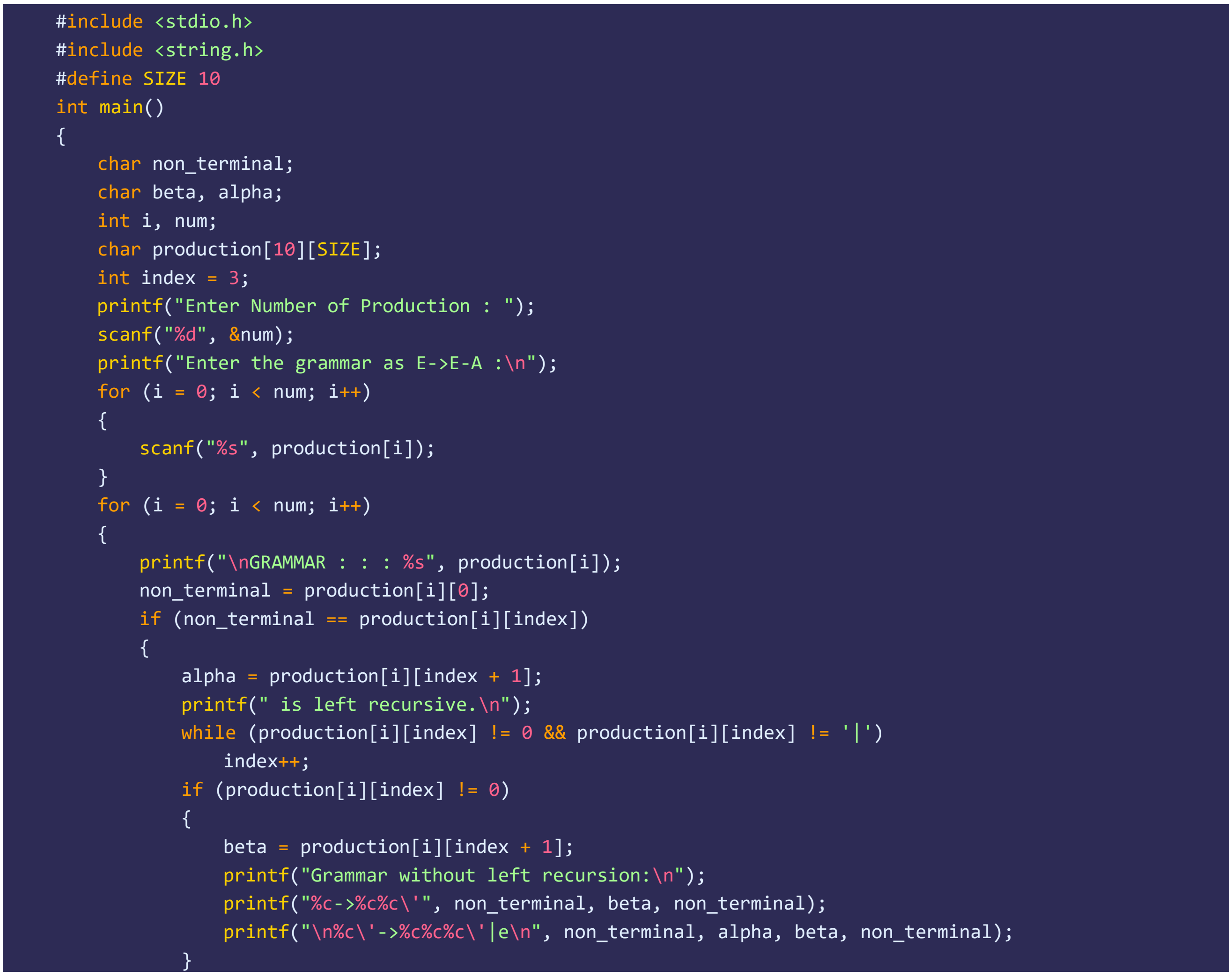
Left Recursion :

AIM: A program for Elimination of Left Recursion.

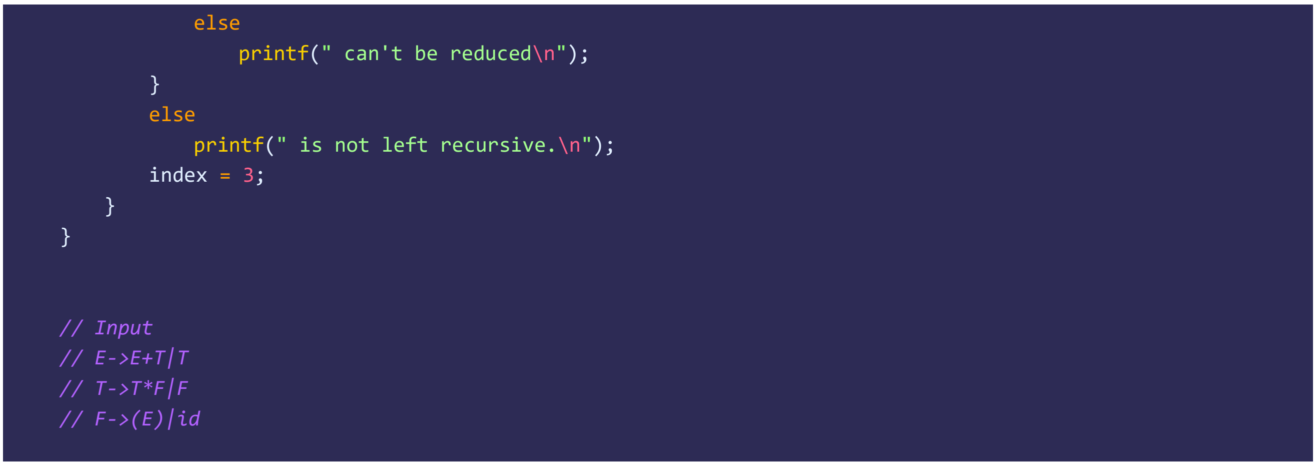
ALGORITHM:

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.

Code:

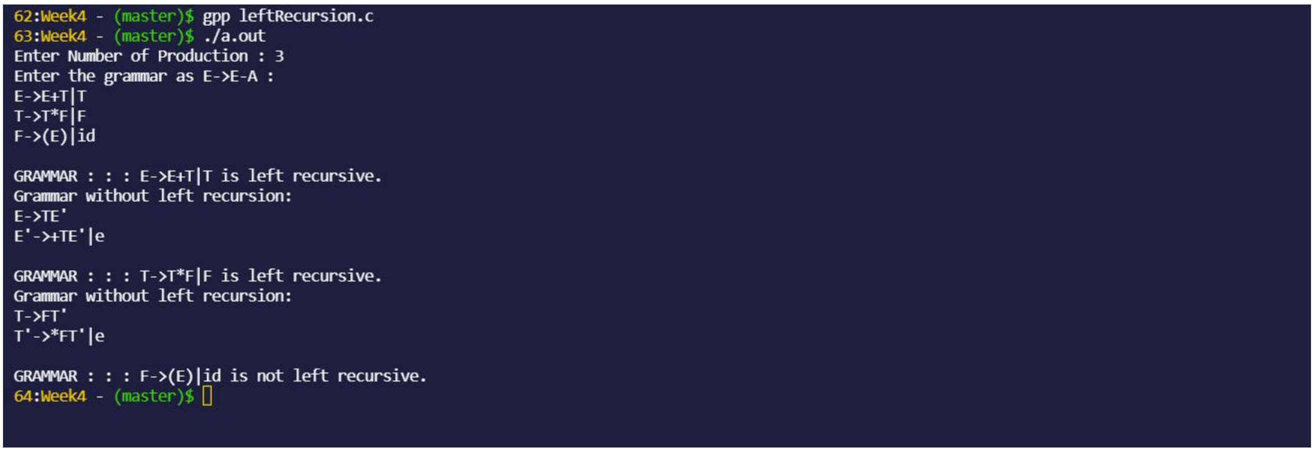


```
#include <stdio.h>
#include <string.h>
#define SIZE 10
int main()
{
    char non_terminal;
    char beta, alpha;
    int i, num;
    char production[10][SIZE];
    int index = 3;
    printf("Enter Number of Production : ");
    scanf("%d", &num);
    printf("Enter the grammar as E->E-A :\n");
    for (i = 0; i < num; i++)
    {
        scanf("%s", production[i]);
    }
    for (i = 0; i < num; i++)
    {
        printf("\nGRAMMAR : : : %s", 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%c\'|e\n", non_terminal, alpha, beta, non_terminal);
            }
```



```
                else
                    printf(" can't be reduced\n");
            }
            else
                printf(" is not left recursive.\n");
            index = 3;
        }
    }


    // Input
    // E->E+T|T
    // T->T*F|F
    // F->(E)|id
```

Output:



```
62:Week4 - (master)$ gpp leftRecursion.c
63:Week4 - (master)$ ./a.out
Enter Number of Production : 3
Enter the grammar as E->E-A :
E->E+T|T
T->T*F|F
F->(E)|id

GRAMMAR : : : E->E+T|T is left recursive.
Grammar without left recursion:
E->TE'
E'->+TE'|e

GRAMMAR : : : T->T*F|F is left recursive.
Grammar without left recursion:
T->FT'
T'->*FT'|e

GRAMMAR : : : F->(E)|id is not left recursive.
64:Week4 - (master)$
```

LEFT FACTORING

AIM: To Write a C Program to eliminate Left Factoring in the given grammar.

ALGORITHM:

- Start
- Get productions from the user
- Check for common left factors in the production
- Group all like productions
- While there are changes to the grammar do

  For each non terminal A do

  Let α be a prefix of maximal length that is shared

  By two or more production choices for A

- If α≠ἐ then

  Let A--> α1|α2|……|αn be all the production choices for A

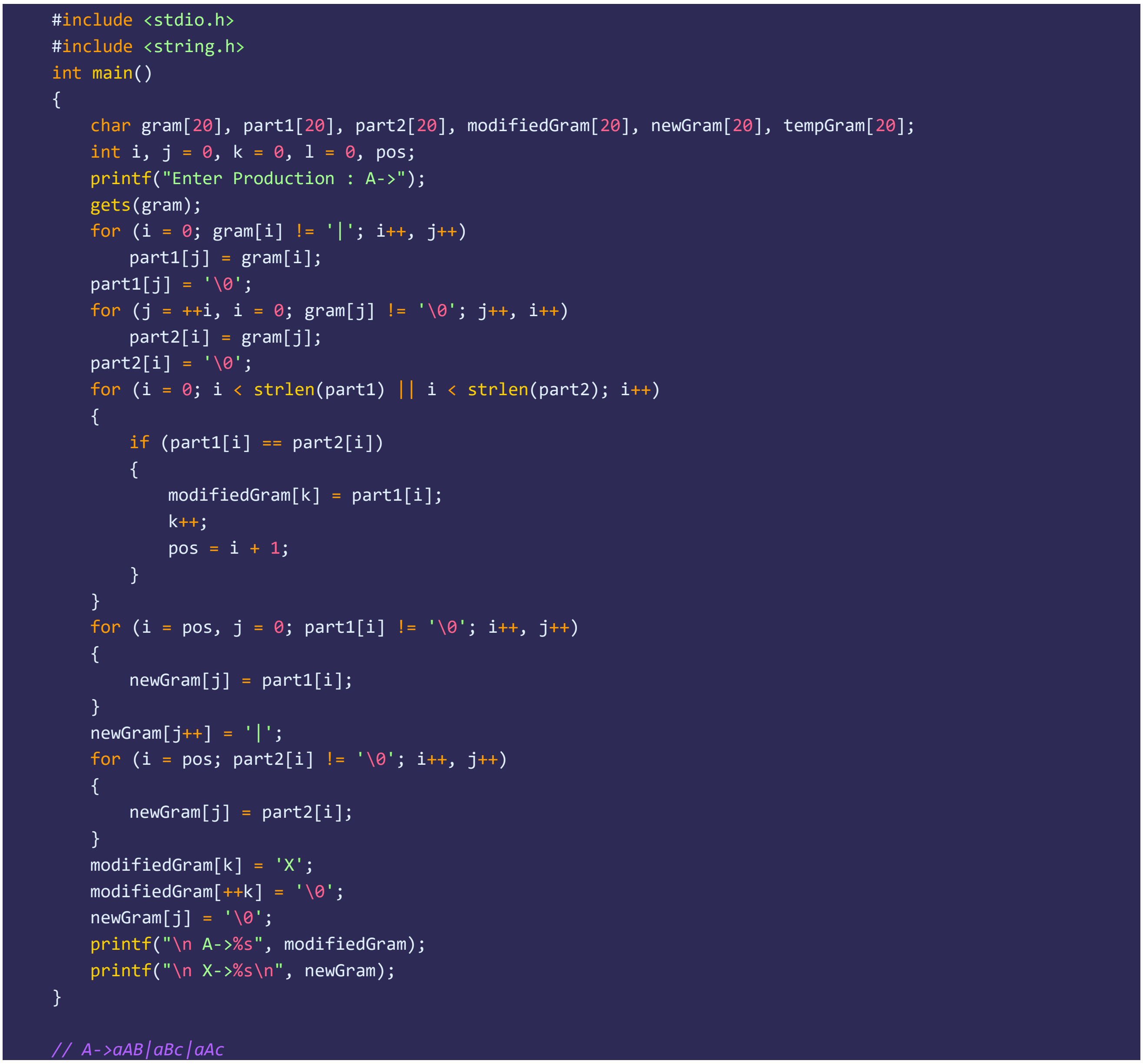
  And suppose that α1,α2,….αk share α ,so that

  A-->αβ1| αβ2| αβ3|……| αβK+1|…|αn,then βj ‘s share

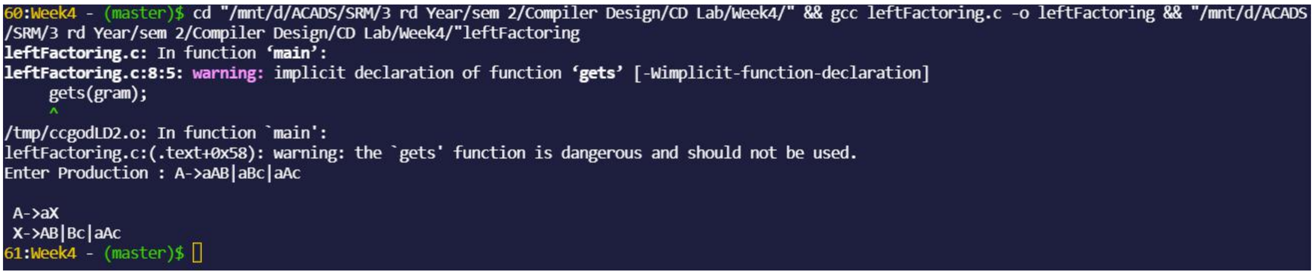
No common prefix ,and αK+1…., αn.. do not share α
Replace the rule A--> α1| α2|….| αn by the rules
A--> αA'| αK+1|..| αn
A'--> β1| β2|..| βK

- Display all productions
- Stop

Code :

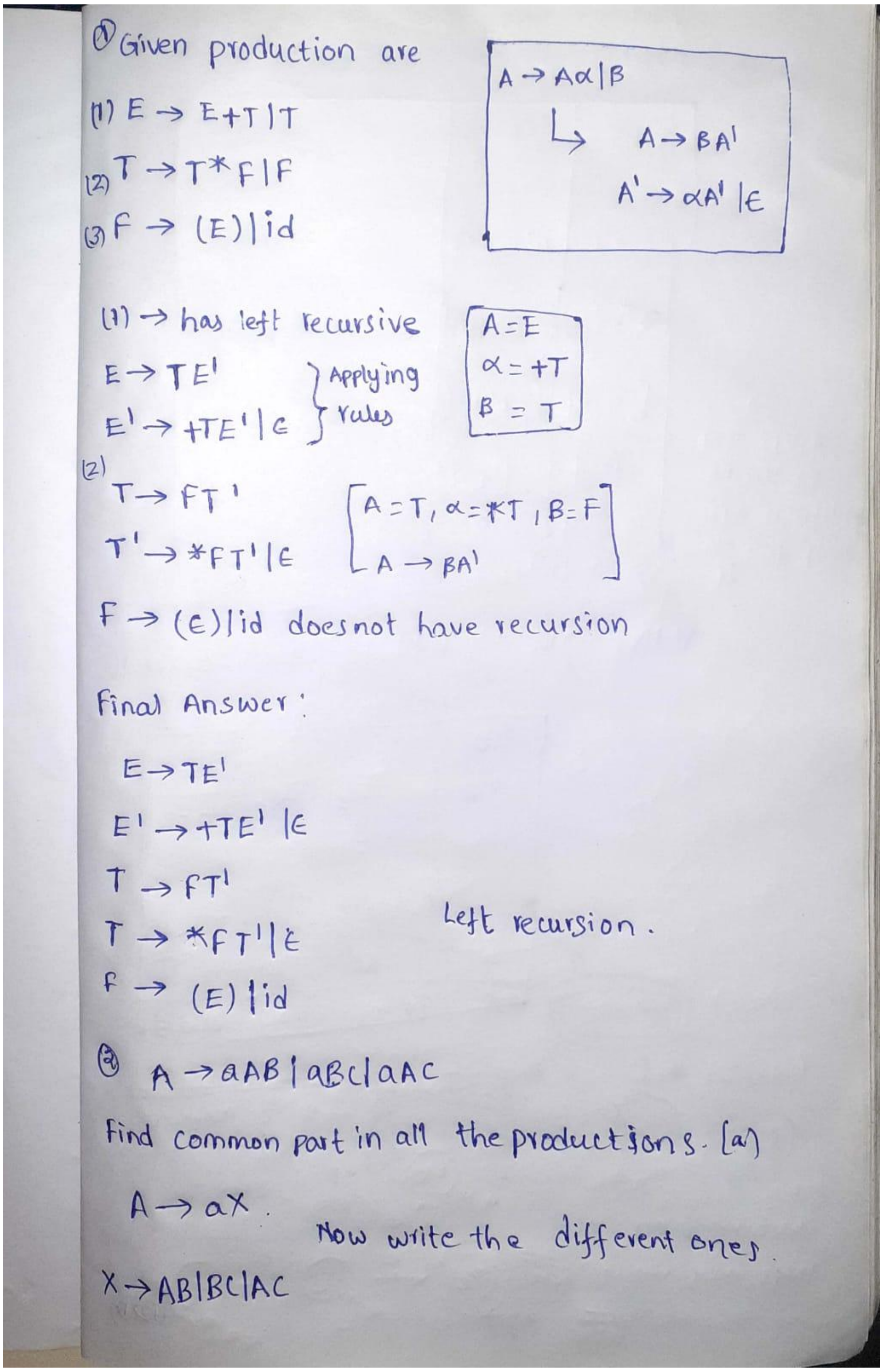


```c
#include <stdio.h>
#include <string.h>
int main()
{
    char gram[20], part1[20], part2[20], modifiedGram[20], newGram[20], tempGram[20];
    int i, j = 0, k = 0, l = 0, pos;
    printf("Enter Production : A->");
    gets(gram);
    for (i = 0; gram[i] != '|'; i++, j++)
        part1[j] = gram[i];
    part1[j] = '\0';
    for (j = ++i, i = 0; gram[j] != '\0'; j++, i++)
        part2[i] = gram[j];
    part2[i] = '\0';
    for (i = 0; i < strlen(part1) || i < strlen(part2); i++)
    {
        if (part1[i] == part2[i])
        {
            modifiedGram[k] = part1[i];
            k++;
            pos = i + 1;
        }
    }
    for (i = pos, j = 0; part1[i] != '\0'; i++, j++)
    {
        newGram[j] = part1[i];
    }
    newGram[j++] = '|';
    for (i = pos; part2[i] != '\0'; i++, j++)
    {
        newGram[j] = part2[i];
    }
    modifiedGram[k] = 'X';
    modifiedGram[++k] = '\0';
    newGram[j] = '\0';
    printf("\n A->%s", modifiedGram);
    printf("\n X->%s\n", newGram);
}

// A->aAB|aBc|aAc
```



```
60:Week4 - (master)$ cd "/mnt/d/ACADS/SRM/3 rd Year/sem 2/Compiler Design/CD Lab/Week4/" && gcc leftFactoring.c -o leftFactoring && "/mnt/d/ACADS
/SRM/3 rd Year/sem 2/Compiler Design/CD Lab/Week4/"leftFactoring
leftFactoring.c: In function ‘main’:
leftFactoring.c:8:5: warning: implicit declaration of function ‘gets’ [-Wimplicit-function-declaration]
     gets(gram);
     ^
/tmp/ccgodLD2.o: In function `main':
leftFactoring.c:(.text+0x58): warning: the `gets' function is dangerous and should not be used.
Enter Production : A->aAB|aBc|aAc

 A->aX
 X->AB|Bc|aAc
61:Week4 - (master)$ 
```

Manual Calculation :



① Given production are

(1) $E \rightarrow E+T \mid T$

(2) $T \rightarrow T*F \mid F$

(3) $F \rightarrow (E) \mid id$

$A \rightarrow A\alpha \mid \beta$

↳ $A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \epsilon$

(1) → has left recursive

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \epsilon$

} Applying rules

$A = E$

$\alpha = +T$

$\beta = T$

(2)

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \epsilon$

[$A = T,\ \alpha = *T,\ \beta = F$

$A \rightarrow \beta A'$]

$F \rightarrow (\epsilon) \mid id$ doesnot have recursion

Final Answer :

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \epsilon$

$T \rightarrow FT'$

$T \rightarrow *FT' \mid \epsilon$

Left recursion .

$F \rightarrow (E) \mid id$

② $A \rightarrow aAB \mid aBC \mid aAC$

Find common part in all the productions. (a)

$A \rightarrow aX$.

Now write the different ones .

$X \rightarrow AB \mid BC \mid AC$