HARSHIT AGGARWAL
RA1911003010782

EXP-4

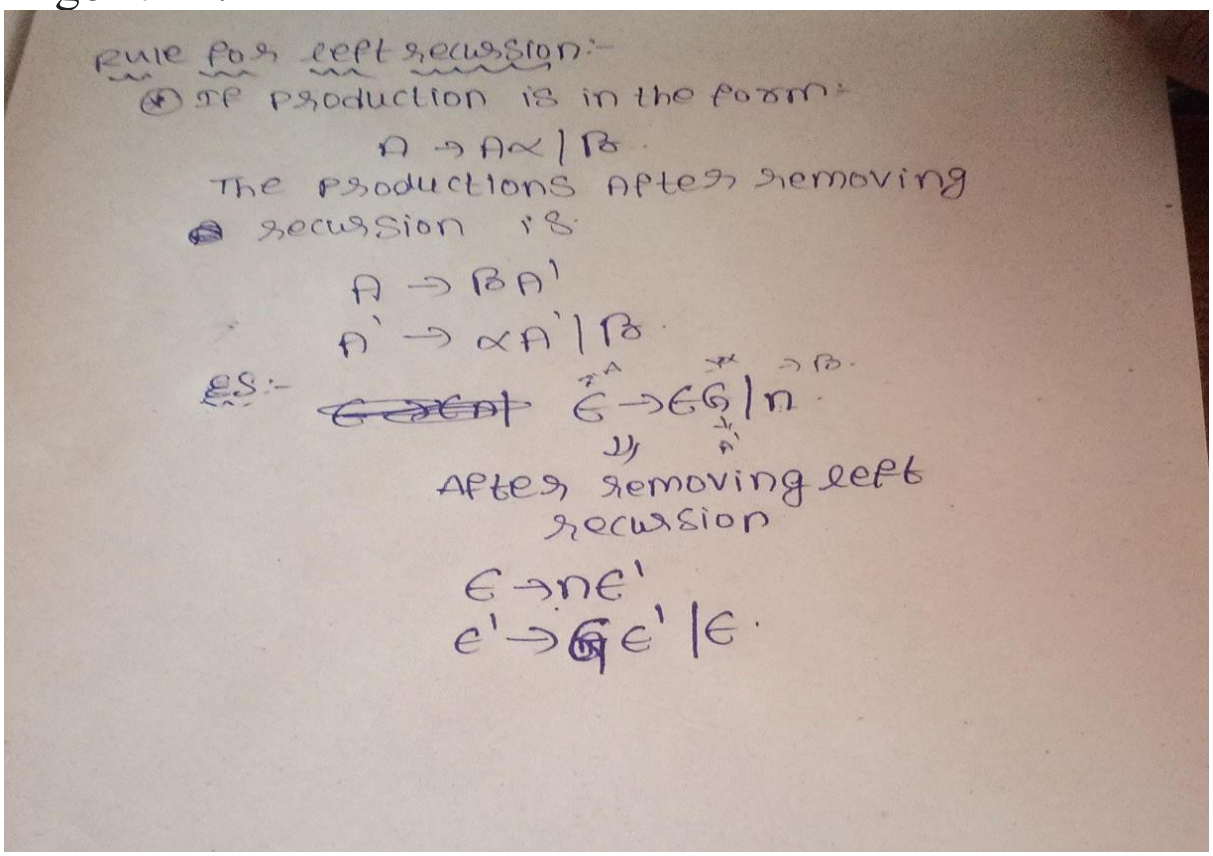
(10/02/2022)

4(a):

AIM: Eliminate Ambiguity from Grammar.
Since Eliminate Ambiguity from Grammar is Undecidable Problem there is no Algorithm form removing Ambiguity.

4(b):

AIM: Eliminate Left Recursion from Grammar.
Algorithm:



Rule for left recursion:-

If production is in the form:-

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

The productions after removing recursion is:

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \beta$.

E.g:- $E \rightarrow EG \mid n$.

After removing left recursion

$E \rightarrow nE'$

$E' \rightarrow GE' \mid \epsilon$.

Code:

```
#include<stdio.h>
```

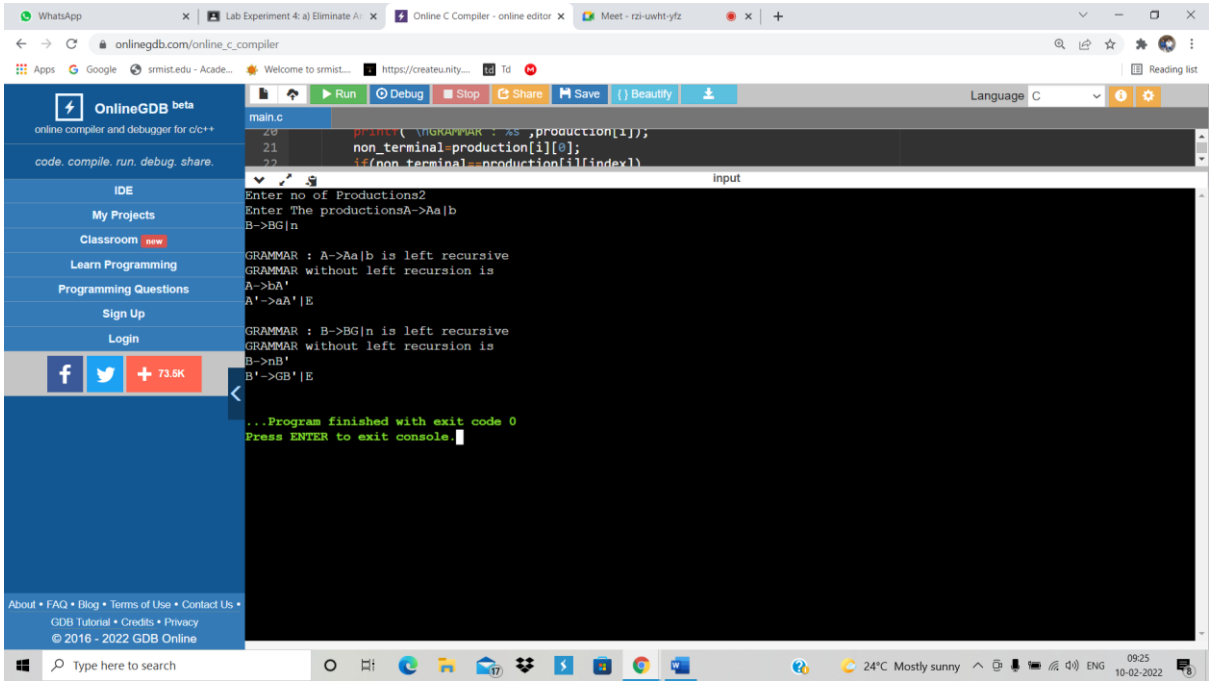
HARSHIT AGGARWAL
RA1911003010782

```
#include<string.h>

#define SIZE 10

int main()

{

    char non_terminal;

    char beta,alpha;

    int num;

    char production[10][SIZE];

    int index=3;

    printf("Enter no of Productions");

    scanf("%d",&num);

    printf("Enter The productions");

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

    {

        scanf("%s",production[i]);

    }

    for(int 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)
```

HARSHIT AGGARWAL
RA1911003010782

```
        {

            beta=production[i][index+1];

            printf("GRAMMAR without left recursion is\n");

            printf("%c->%c%c\'",non_terminal,beta,non_terminal);

            printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);

        }

       else

        printf("Canoot be reduced");

      }

      else

        printf(" is not left Recursive\n");

      index=3;

    }

}
```

OUTPUT:



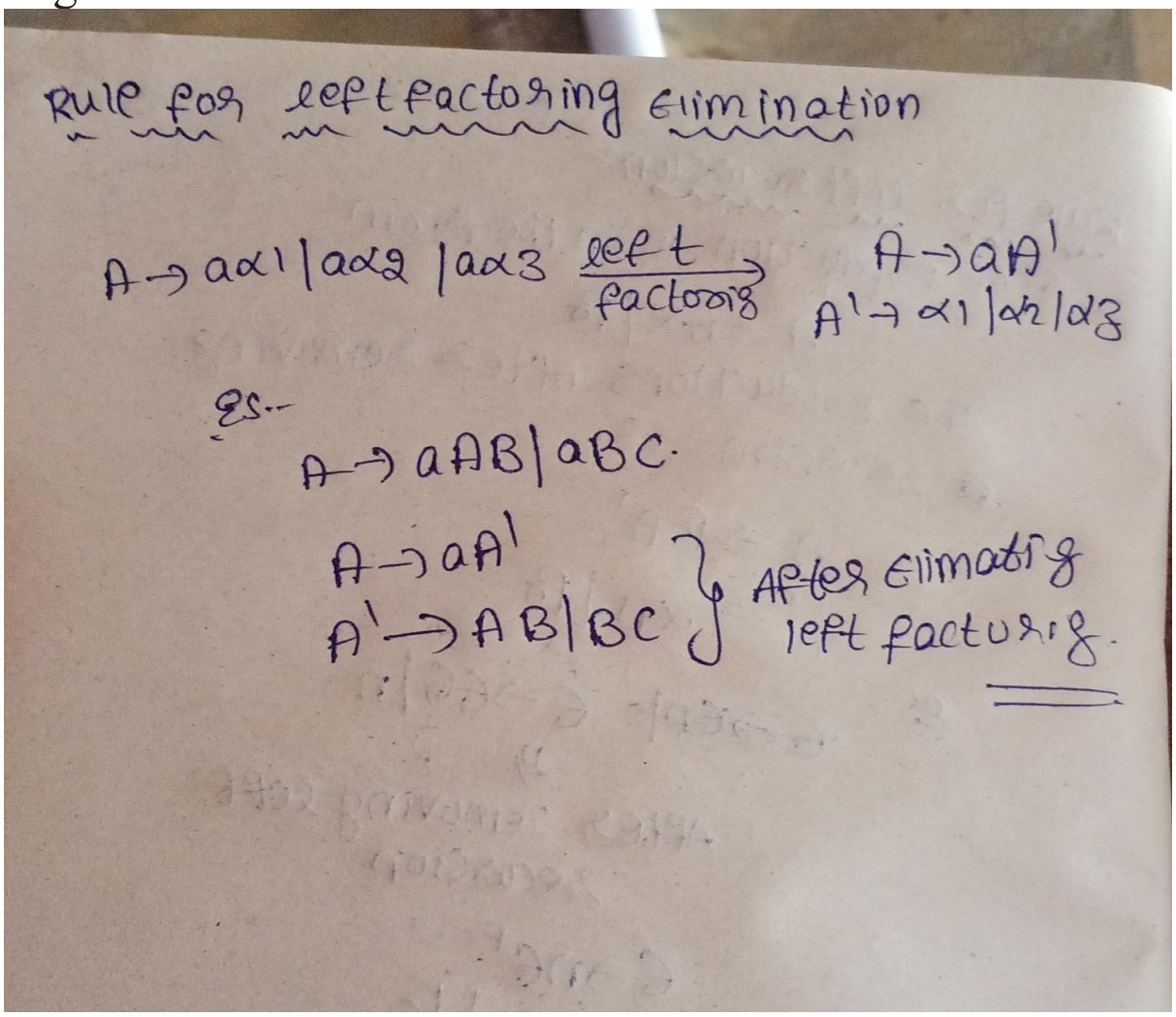
RESULT : Hence Elimination of Left Recursion is Completed.

HARSHIT AGGARWAL
RA1911003010782

4.C:

Aim: To Eliminate Left Factoring

Algorithm:



Rule for left factoring elimination

$A \to a\alpha_1 | a\alpha_2 | a\alpha_3 \xrightarrow[\text{factoring}]{\text{left}} A \to aA'$, $A' \to \alpha_1 | \alpha_2 | \alpha_3$

Ex:-

$A \to aAB | aBC$.

$A \to aA'$
$A' \to AB | BC$ } After eliminating left factoring.

CODE:

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int main()

{

    char gram[20],part1[20],part2[20],modifiedgram[20];

    char newgram[20],tempgram[20];
```

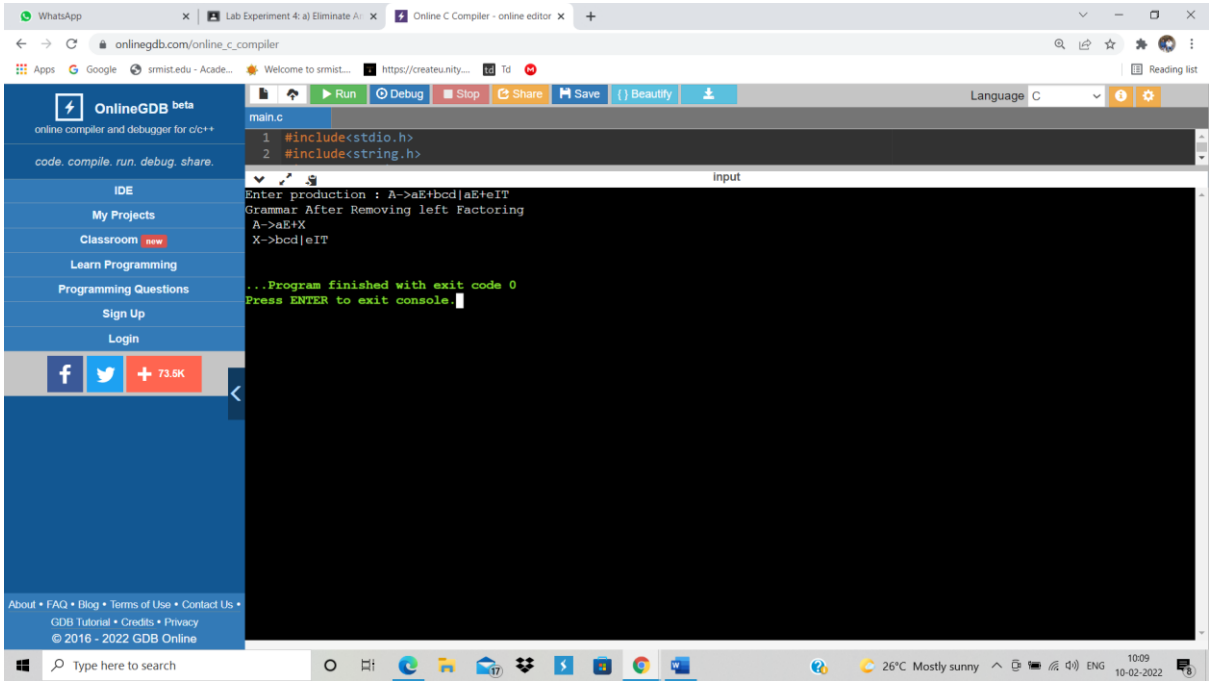
HARSHIT AGGARWAL
RA1911003010782

```
int i,j=0,k=0,l=0,pos;

printf("Enter production : A->");

scanf("%s",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';
```

HARSHIT AGGARWAL
RA1911003010782

```
    modifiedgram[++k]='\0';

    newgram[j]='\0';

   printf("Grammar After Removing left Factoring");

   printf("\n A->%s",modifiedgram);

   printf("\n X->%s\n",newgram);



}
```

OUTPUT:



RESULT: Hence Left Factoring is Eliminated