Name: Aashutosh Kumar Pandit
Roll No: CH.EN.U4CSE22076

Lab-3

1. AIM: To implement eliminate left recursion and left factoring from the given grammar using C program.

Left Factoring:

CODE:

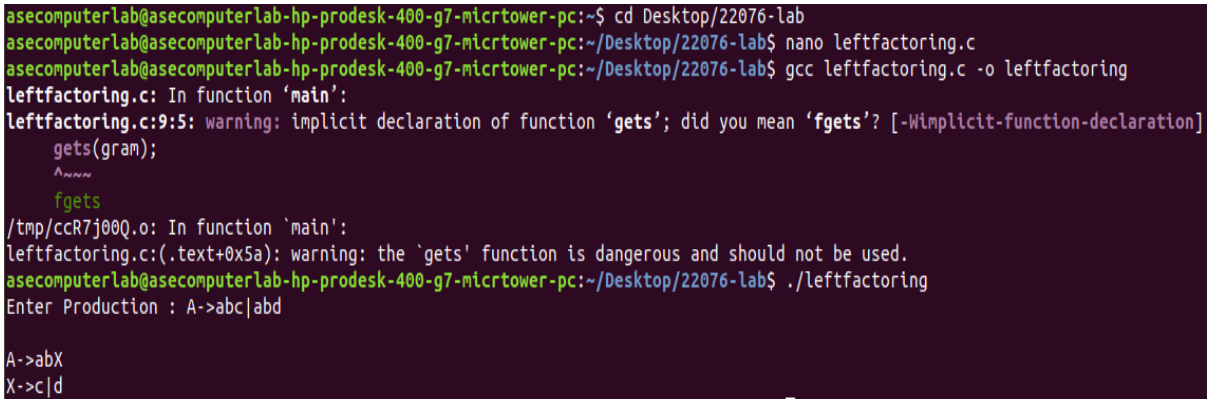
leftfactoring.c

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

int main() {
   char gram[100], part1[100], part2[100], modifiedGram[100],newGram[100];
   int i, j = 0, k = 0, pos = 0;

   printf("Enter Production : A->");
   gets(gram);

   // Split input into part1 and part2 at '|'
   for(i = 0; gram[i] != '|' && gram[i] != '\0'; i++, j++)
      part1[j] = gram[i];
   part1[j] = '\0';

   j++;  // skip '|'
   for(i = j, j = 0; gram[i] != '\0'; i++, j++)
      part2[j] = gram[i];
```

```
part2[j] = '\0';

// Find longest common prefix
for(i = 0; i < strlen(part1) && i < strlen(part2); i++) {
    if(part1[i] == part2[i]) {
        modifiedGram[k++] = part1[i];
        pos = i + 1;  // position after common prefix
    } else {
        break;
    }
}

// Construct newGram with suffixes after the common prefix
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];
newGram[j] = '\0';

// Append 'X' to modifiedGram
modifiedGram[k++] = 'X';
modifiedGram[k] = '\0';

// Print the result
printf("\nA->%s", modifiedGram);
printf("\nX->%s\n", newGram);
```

```
    return 0;
}
```

OUTPUT:



LEFT RECURSION

AIM: To implement left recursion using C.

CODE:

Leftrecursion.c

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

#define SIZE 100

int main() { char non_terminal; char beta[SIZE], alpha[SIZE]; int num; char
production[10][SIZE]; int index;

printf("Enter Number of Productions: ");
scanf("%d", &num);
printf("Enter the grammar productions (e.g. E->E-A):\n");

// Read grammar 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];
   index = 3; // index where RHS starts after "->"

   // Check if production is left recursive
   if(production[i][index] == non_terminal) {
      printf(" is left recursive.\n");

      // Extract alpha (the part after the non_terminal in left recursion)
      int alpha_idx = 0;
      index++; // move past non_terminal on RHS
      while(production[i][index] != '\0' && production[i][index] != '|') {
         alpha[alpha_idx++] = production[i][index++];
```

```
        }
        alpha[alpha_idx] = '\0';

        // Check if there is '|' to separate beta
        if(production[i][index] == '|') {
            index++; // move past '|'

            // Extract beta (the part after '|')
            int beta_idx = 0;
            while(production[i][index] != '\0') {
                beta[beta_idx++] = production[i][index++];
            }
            beta[beta_idx] = '\0';

            // Print grammar without left recursion
            printf("Grammar without left recursion:\n");
            printf("%c->%s%c'\n", non_terminal, beta, non_terminal);
            printf("%c'->%s%c'|epsilon\n", non_terminal, alpha, non_terminal);
        } else {
            printf("Cannot be reduced (no alternative beta found).\n");
        }
    } else {
        printf(" is not left recursive.\n");
    }
}

return 0;


}
```

Output:

```
asecomputerlab@asecomputerlab-hp-prodesk-400-g7-micrtower-pc:~/Desktop/22076-lab$ nano left_recursion.c
asecomputerlab@asecomputerlab-hp-prodesk-400-g7-micrtower-pc:~/Desktop/22076-lab$ gcc left_recursion.c -o left_recursion
asecomputerlab@asecomputerlab-hp-prodesk-400-g7-micrtower-pc:~/Desktop/22076-lab$ ./left_recursion
Enter Number of Productions: 2
Enter the grammar productions (e.g. E->E-A):
E->E-A
E->b

GRAMMAR: E->E-A is left recursive.
Cannot be reduced (no alternative beta found).

GRAMMAR: E->b is not left recursive.
```

```
asecomputerlab@asecomputerlab-hp-prodesk-400-g7-micrtower-pc:~/Desktop/22076-lab$ ./left_recursion
Enter Number of Productions: 2
Enter the grammar productions (e.g. E->E-A):
E->EA|A
A->A|B

GRAMMAR: E->EA|A is left recursive.
Grammar without left recursion:
E->AE'
E'->AE'|epsilon

GRAMMAR: A->A|B is left recursive.
Grammar without left recursion:
A->BA'
A'->A'|epsilon
```