Date: 21/07/2025

Experiment No: 02

Aim: To eliminate left recursion and left factoring from the given grammar using C program.

Code:

Left Factoring:

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

int main() {
    char gram[100], part1[50], part2[50];
    char modifiedGram[50], newGram[50];
    int i, j = 0, k = 0, l = 0, pos = 0;

    printf("Enter Production (e.g., A->abc|abd): ");
    fgets(gram, sizeof(gram), stdin);

    // Remove newline character
    gram[strcspn(gram, "\n")] = 0;

    // Skip "A->" (assumed fixed)
    int start = 3;

    // Split production at '|'
    for (i = start; 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';

    // Find common prefix
    for (i = 0; part1[i] != '\0' && part2[i] != '\0'; i++) {
        if (part1[i] == part2[i]) {
            modifiedGram[k++] = part1[i];
            pos = i + 1;
        } else {
            break;
        }
    }
```

```
    modifiedGram[k++] = 'X';
    modifiedGram[k] = '\0';

    // Get the remaining parts after 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';

    printf("\nLeft Factored Grammar:\n");
    printf("A->%s\n", modifiedGram);
    printf("X->%s\n", newGram);

    return 0;
}
```

Output:

```
asecomputerlab@hp-desktop:~/Desktop/22075$ gcc 2p.c
asecomputerlab@hp-desktop:~/Desktop/22075$ ./a.out
Enter Production (e.g., A->abc|abd): A->aE+bcD |aE+eIT

Left Factored Grammar:
A->aE+X
X->bcD |eIT
```

Left recursion:

Aim: To implement left recursion using C.

Code:

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

#define SIZE 10

int main() {
    char non_terminal;
    char alpha[SIZE], beta[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 or E->Ea|b):\n");
    for (int i = 0; i < num; i++) {
        scanf("%s", production[i]);
    }

    for (int i = 0; i < num; i++) {
        printf("\nGRAMMAR: %s", production[i]);
        index = 3;  // Start after 'A->'
        non_terminal = production[i][0];

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

            int j = index + 1, k = 0;
            while (production[i][j] != '|' && production[i][j] != '\0') {
                alpha[k++] = production[i][j++];
            }
            alpha[k] = '\0';
```

```
            if (production[i][j] == '|') {
                j++; // skip '|'
                k = 0;
                while (production[i][j] != '\0') {
                    beta[k++] = production[i][j++];
                }
                beta[k] = '\0';

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

    return 0;
}
```

Output:

```
asecomputerlab@hp-desktop:~/Desktop/22075$ gcc left_recursion.c
asecomputerlab@hp-desktop:~/Desktop/22075$ ./a.out
Enter Number of Productions: 1
Enter the grammar productions (e.g., E->E-A or E->Ea|b):
E->Ea|b

GRAMMAR: E->Ea|b is left recursive.
Grammar without left recursion:
E -> bE'
E' -> aE' | ε
```

Conclusion:

The program to implement left factoring and left recursion has been successfully executed.