Name: Nimish Kushwaha

Reg.No.:CH.EN.U4CSE22073

Experiment No: 02

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

i. Left factoring

Code:

```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);  // Note: unsafe, consider fgets for real code

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

    for (j = i + 1, i = 0; gram[j] != '\0'; j++, i++)
        part2[i] = gram[j];
    part2[i] = '\0';

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

    // Build new production after factoring
    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';  // new variable for factoring
    modifiedGram[k] = '\0';
    newGram[j] = '\0';

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

    return 0;
}
```

Output:

```
ubuntu:~$ gcc ex2.c
ex2.c: In function ‘main’:
ex2.c:9:5: warning: implicit declaration of function ‘gets’; did you mean ‘fgets
’? [-Wimplicit-function-declaration]
     gets(gram);  // Note: unsafe, consider fgets for real code
     ^~~~
     fgets
/tmp/ccnIWJvK.o: In function `main':
ex2.c:(.text+0x5a): warning: the `gets' function is dangerous and should not be
used.
ubuntu:~$ ./a.out
Enter Production : A->aE+X

 A->X
 X->aE+X|
ubuntu:~$
```

ii. **Left Recursion**

Code:

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

#define SIZE 100

int main() {
    char non_terminal;
    char beta, alpha;
    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");
    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; // position after '->'

        if (production[i][index] == non_terminal) {
            alpha = production[i][index + 1];
            printf(" is left recursive.\n");

            // Move index forward to the end of alpha part (before '|')
            while (production[i][index] != '\0' && production[i][index] != '|') {
                index++;
            }
```

```
                }

                if (production[i][index] == '|') {
                    beta = production[i][index + 1];
                    printf("Grammar without left recursion:\n");
                    printf("%c->%c%c'\n", non_terminal, beta, non_terminal);
                    printf("%c'->%c%c'|ε\n", non_terminal, alpha, non_terminal);
                } else {
                    printf(" can't be reduced\n");
                }
            } else {
                printf(" is not left recursive.\n");
            }
        }

        return 0;
    }
```

Output:

```
ubuntu:~$ gedit lab2.1.c
ubuntu:~$ gcc lab2.1.c
ubuntu:~$ ./a.out
Enter Number of Productions: 2
Enter the grammar productions (e.g. E->E-A):
E->A/B
eX+B

GRAMMAR: E->A/B is not left recursive.

GRAMMAR: eX+B is not left recursive.
ubuntu:~$
```

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