Exercise 3
Elimination of Immediate Left Recursion

Nanda H Krishna
312217104093

January 31, 2020

1 C Program

```
#include<stdio.h>
#include<string.h>
int main()
{
    char non_terminal, productions[10][100], splits[10][10];
    int num;
    printf("Enter number of productions: ");
    scanf("%d", &num);
    printf("Enter the grammar:\n");
    for(int i = 0; i < num; i++)
        scanf("%s", productions[i]);
    for(int i = 0; i < num; i++)
    {
        printf("\n%s", productions[i]);
        non_terminal = productions[i][0];
        char production[100], *token;
        int j, flag = 0;
        for(j = 0; productions[i][j + 3] != '\0'; j++)
            production[j] = productions[i][j + 3];
        production[j] = '\0'; j = 0;
        token = strtok(production, "|");
        while(token != NULL)
        {
            strcpy(splits[j], token);
            if(token[0] == non_terminal && flag == 0)
                flag = 1;
            else if(token[0] != non_terminal && flag == 1)
                flag = 2;
            j++;
            token = strtok(NULL, "|");
        }
        if(flag == 0)
            printf(" is not left recursive.\n");
        else if(flag == 1)
            printf(" is left recursive, cannot reduce.\n");
        else
        {
            printf(" is left recursive. After elimination:\n");
            flag = 0;
            for(int k = 0; k < j; k++)
            {
                if(splits[k][0] != non_terminal) {
                    if(flag != 0)
                    {
                        printf("|%s%c\'", splits[k], non_terminal);
                    }
                    else
                    {
                        flag = 1;
                        printf("%c->%s%c\'", non_terminal,
                                        splits[k], non_terminal);
                    }
                }
            }
            printf("\n");
            flag = 0;
            for(int k = 0; k < j; k++)
            {
                if(splits[k][0] == non_terminal) {
                    if(flag != 0)
                    {
                        printf("|%s%c\'", splits[k] + 1, non_terminal);
                    }
                    else
                    {
                        flag = 1;
                        printf("%c\'->%s%c\'", non_terminal,
                                        splits[k] + 1, non_terminal);
                    }
                }
            }
            printf("|e\n");
        }
    }
}
```

2 Input Grammar

2.1 Example 1

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow id \mid (E) \end{aligned}$$

2.2 Example 2

$$\begin{aligned} A &\rightarrow A\alpha_1 \mid A\alpha_2 \mid \beta_1 \mid \beta_2 \mid B \\ B &\rightarrow B\gamma \end{aligned}$$

3 Expected Output

3.1 Example 1

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \epsilon \\ F &\rightarrow id \mid (E) \end{aligned}$$

3.2 Example 2

$$\begin{aligned} A &\rightarrow \beta_1 A' \mid \beta_2 A' \mid BA' \\ A' &\rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \epsilon \\ B &\rightarrow B\gamma \text{ (cannot eliminate)} \end{aligned}$$

4 Output

4.1 Example 1

```
Enter number of productions: 3
Enter the grammar:
E->E+T|T
T->T*F|F
F->id|(E)

E->E+T|T is left recursive. After elimination:
```

```
E->TE'
E'->+TE'|e

T->T*F|F is left recursive. After elimination:
T->FT'
T'->*FT'|e

F->id|(E) is not left recursive.
```

4.2 Example 2

```
Enter number of productions: 2
Enter the grammar:
A->Aa1|Aa2|b1|b2|B
B->Bc

A->Aa1|Aa2|b1|b2|B is left recursive. After elimination:
A->b1A'|b2A'|BA'
A'->a1A'|a2A'|e

B->Bc is left recursive, cannot reduce.
```