Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VI

20 February 2021

UCS 1602 - Compiler Design

Exercise 3: Elimination of Left Recursion Using C

Aim:

Write a program in C to find whether the given grammar is **Left Recursive** or not. If it is found to be left recursive, convert the grammar in such a way that the left recursion is removed.

Code:

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

int main(){
    /*
        Sample Input Format:    E->E+T|T
                                T->T*F|F
                                F->i
    */

    char productions[100][100], sub_prods[100][100];
    char non_terminal;
    int num_prods, i, j, k, flag = 0;

    printf("\n\t\tElimination of Left Recursion\n");
    printf("\nEnter the number of Productions: ");
    scanf("%d", &num_prods);

    printf("\nEnter the Grammar:\n");

    for(i = 0; i < num_prods; i++){
        //Getting Input
        scanf("%s", productions[i]);
    }

    printf("\nGiven Grammar:\n");

    for(i = 0; i < num_prods; i++){
        //Printing the Grammar, and checking for left recursions
        printf("%s\n", productions[i]);

        if(productions[i][0] == productions[i][3]){
            flag = 1;
        }
    }

    if(flag == 0){
        //If Grammar is not left recursive, exit
        printf("\nGrammar is not Left Recursive.");
        return 0;
    }

    //Otherwise, Grammar is left recursive, parse and remove it
    printf("\nGrammar is Left Recursive.");
    printf("\n\nGrammar after removal of Left Recursion:");

```

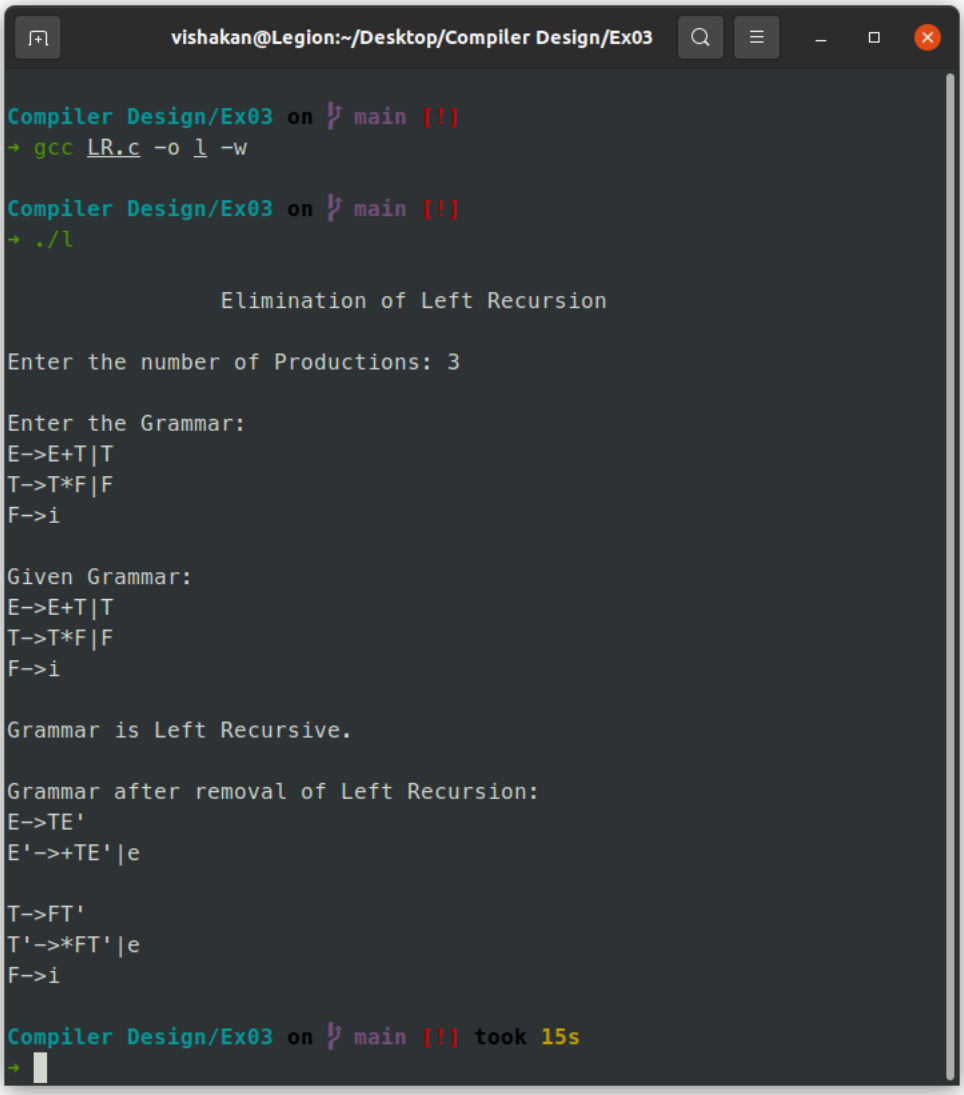
```
48      for(i = 0; i < num_prods; i++){
49          //Parse each production one by one
50          non_terminal = productions[i][0];
51
52          char *split, production[100];
53          flag = 0;
54
55          //Store the RHS of the production alone
56          for(j = 0; productions[i][j + 3] != '\0'; j++){
57              production[j] = productions[i][j + 3];
58          }
59
60          production[j] = '\0';
61          j = 0;
62
63          //Split at the sub-expression level when there is an OR operator
64          split = strtok(production, "|");
65
66          while(split != NULL){
67              //Store the subexpression in a new productions array
68              strcpy(sub_prods[j], split);
69
70              if(split[0] == non_terminal && flag == 0){
71                  //Seeing an immediate left recursion, with no other
    productions
72                  //for the same non-terminal
73                  //This type of Left Recursion cannot be removed
74                  flag = 1;
75              }
76              else if(split[0] != non_terminal && flag == 1){
77                  //Already seen a left recursion, but now we have seen
78                  //another production with some terminal symbol
79                  //for the same non-terminal
80                  flag = 2;
81              }
82
83              j++;
84              split = strtok(NULL, "|");
85              //split and loop till all productions are parsed
86          }
87
88          if(flag != 2){
89              //flag == 0 => no LR
90              //flag == 1 => LR of the form A->Ab which cannot be removed
91              printf("%s\n", productions[i]);
92          }
93
94          if(flag == 2){
95              //Remove the left recursion if there's another production with
     terminal symbol
96              printf("\n");
```

```
97              flag = 0;
98
99              for(k = 0; k < j; k++){
100                 if(sub_prods[k][0] != non_terminal){
101                     //Loop until the non-terminal causing the LR is not
    found, for 1st production rule
102                     if(flag != 0){
103                         //Removed the LR by starting with the other non-
    terminal/ID,
104                         //thus add the remaining sub-productions
105                         printf("|%s%c\'", sub_prods[k], non_terminal);
106                     }
107                     else{
108                         //No left recursion with that particular sub-
    production
109                         //thus make it as a new production with a new non-
    terminal
110                         flag = 1;
111                         printf("%c->%s%c\'", non_terminal, sub_prods[k],
    non_terminal);
112                     }
113                 }
114             }
115             printf("\n");
116             flag = 0;
117
118             for(k = 0; k < j; k++){
119                 if(sub_prods[k][0] == non_terminal){
120                     //Loop until the non-terminal causing the LR is found,
     for 2nd production rule
121                     if(flag != 0){
122                         //Add the remaining sub-productions, since the LR
    has been removed
123                         printf("|%s%c\'", sub_prods[k] + 1, non_terminal);
124                     }
125                     else{
126                         //k sub-production contains the LR causing term,
    thus first print the
127                         //next sub-production followed by a new non-
    terminal as a new production
128                         //2D Array Manipulation, sub_prods[k] + 1
    essentially prints
129                         //the string sub_prods[k][1] till sub_prods[k][n]
130                         flag = 1;
131                         printf("%c\'->%s%c\'", non_terminal, sub_prods[k]
    + 1, non_terminal);
132                     }
133                 }
134             }
135             printf("|e\n");
136         }
```

```
137
138     }
139
140
141     return 0;
142 }
143
144 /*
145 OUTPUT:
146
147 gcc LR.c -o l -w
148 ./l
149
150         Elimination of Left Recursion
151
152 Enter the number of Productions: 3
153
154 Enter the Grammar:
155 E->E+T|T
156 T->T*F|F
157 F->i
158
159 Given Grammar:
160 E->E+T|T
161 T->T*F|F
162 F->i
163
164 Grammar is Left Recursive.
165
166 Grammar after removal of Left Recursion:
167 E->TE'
168 E'->+TE'|e
169
170 T->FT'
171 T'->*FT'|e
172 F->i
173
174 */
```

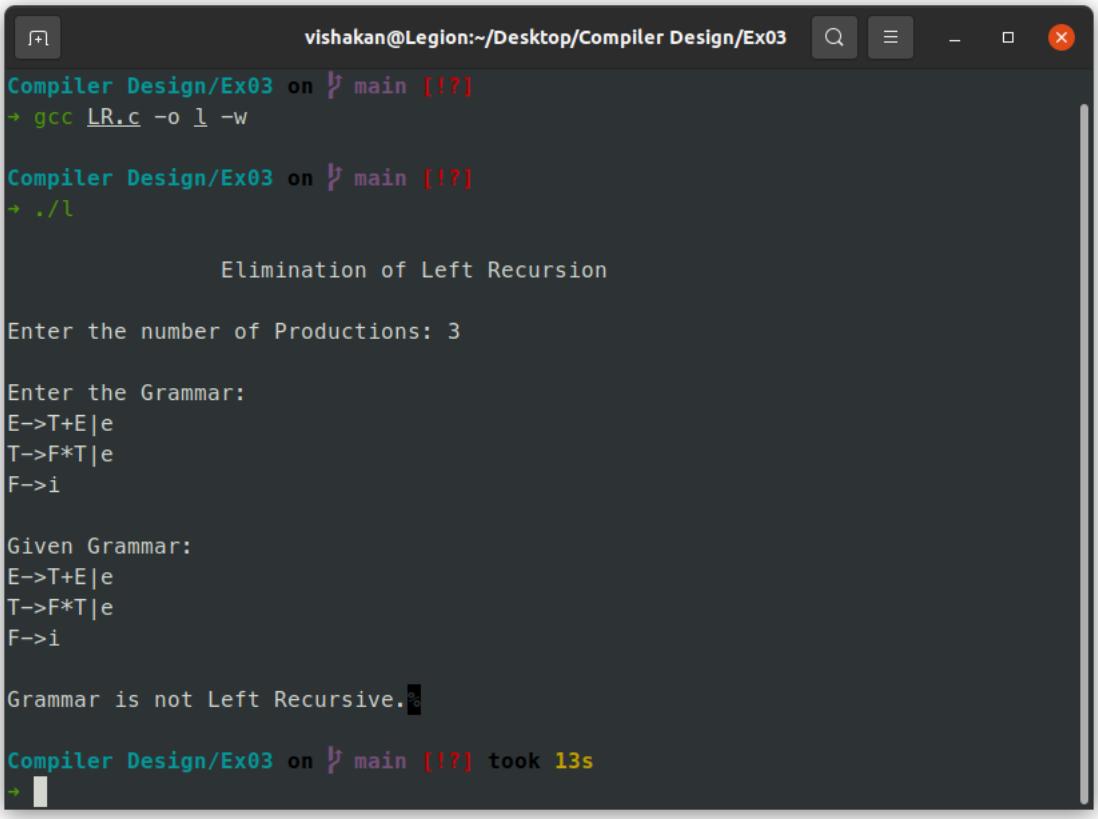
Output - Left Recursive Grammar:

Figure 1: Console Output for a Left Recursive Grammar.



Output - Non Left Recursive Grammar:

Figure 2: Console Output for a Non Left Recursive Grammar.



Learning Outcome:

- I understood about left recursive grammars.
- I understood the need for this type of conversion, as top-down parsers cannot handle left recursive grammars.
- I was able to perform a check of whether or not a grammar is left recursive using C.
- I implemented a conversion in C which converts left recursive grammar to non left recursive grammar.
- I refreshed my 2D-char array manipulation concepts in C.