



Premier University Chittagong

Department of Computer Science and Engineering

Course Title : Compiler Construction Lab

Course Code: CSE 454

Report No : 07

Report Title : Write a Program to Perform Left Factoring on a Given Grammar and Show the Factored Outpu

Submission Date: 24.09.2025

Submitted By

Name : Rayanul Kader Chowdhury Abid
ID : 210401020 2162
Semester : 8th A Section

Submitted To

Ms. Tanni Dhoom
Assistant Professor
Department of Computer Science and Engineering

Experiment No: 7

Experiment Name: Write a program to perform left factoring on a given grammar and show the factored output.

Objectives:

To design and implement a program that detects common prefixes in grammar productions and rewrites the grammar using left factoring. Left factoring is necessary to remove ambiguity and make the grammar suitable for top-down parsing.

Algorithm

1. Input the non-terminal and its productions from the user.
2. For the given set of productions:
 - Compare all pairs of productions to find the longest common prefix.
 - Count how many productions share this prefix.
3. If a common prefix exists:
 - Create a new non-terminal by appending a prime symbol (' \prime ') to the original non-terminal.
 - Rewrite the original productions as:
$$A \rightarrow \text{prefix } A'$$
$$A' \rightarrow \text{remaining_parts} \mid \epsilon$$
where remaining_parts are the suffixes of productions after removing the prefix.
4. Recursively apply left factoring to the newly created non-terminal and any remaining productions.
5. Print the factored grammar after processing all productions.

Code:

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

```

#define MAX 100

int ntCount = 0;

void newNonTerminal(char base[], char result[]) {
    strcpy(result, base);
    for (int i = 0; i <= ntCount; i++) {
        strcat(result, "'");
    }
    ntCount++;
}

void findLongestPrefix(int n, char prods[MAX][MAX], char prefix[MAX], int *count) {
    prefix[0] = '\0';
    *count = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            int k = 0;
            while (prods[i][k] && prods[j][k] && prods[i][k] == prods[j][k]) {
                k++;
            }
            if (k > 0) {
                char temp[MAX];
                strncpy(temp, prods[i], k);
                temp[k] = '\0';
                int c = 0;
                for (int p = 0; p < n; p++) {
                    if (strcmp(prods[p], temp, strlen(temp)) == 0)
                        c++;
                }
            }
        }
    }
}

```

```

if (c > 1 && strlen(temp) > strlen(prefix)) {
    strcpy(prefix, temp);
    *count = c;
}
}

void leftFactoring(char nonTerminal[], int n, char productions[MAX][MAX]) {
    char prefix[MAX];
    int prefixCount;
    findLongestPrefix(n, productions, prefix, &prefixCount);
    if (strlen(prefix) == 0) {
        printf("%s -> ", nonTerminal);
        for (int i = 0; i < n; i++) {
            printf("%s", productions[i]);
            if (i != n - 1) printf(" | ");
        }
        printf("\n");
        return;
    }
    char newNT[MAX];
    newNonTerminal(nonTerminal, newNT);
    printf("%s -> %s%s\n", nonTerminal, prefix, newNT);
    char newProds[MAX][MAX];
    int newCount = 0;
    char remaining[MAX][MAX];
    int remCount = 0;
    for (int i = 0; i < n; i++) {
        if (strncmp(productions[i], prefix, strlen(prefix)) == 0) {
            if (strlen(productions[i]) == strlen(prefix))
                strcpy(newProds[newCount++], "epsilon");
        }
    }
}

```

```

else
strcpy(newProds[newCount++], productions[i] + strlen(prefix));
} else {
strcpy(remaining[remCount++], productions[i]);
}}
leftFactoring(newNT, newCount, newProds);
if (remCount > 0) {
leftFactoring(nonTerminal, remCount, remaining);
}}
int main() {
char nonTerminal[MAX];
int n;
char productions[MAX][MAX];
printf("Enter the non-terminal: ");
scanf("%s", nonTerminal);
printf("Enter the number of productions: ");
scanf("%d", &n);
getchar();
printf("Enter the productions:\n");
for (int i = 0; i < n; i++) {
fgets(productions[i], MAX, stdin);
productions[i][strcspn(productions[i], "\n")] = '\0';
}
printf("\nOriginal Grammar:\n%s -> ", nonTerminal);
for (int i = 0; i < n; i++) {
printf("%s", productions[i]);
if (i != n - 1) printf(" | ");
}
}

```

```
printf("\n\nFactored Grammar:\n");
leftFactoring(nonTerminal, n, productions);
return 0;
}
```

4 Input:

Enter the non-terminal: A

Enter the number of productions: 4

Enter the productions:

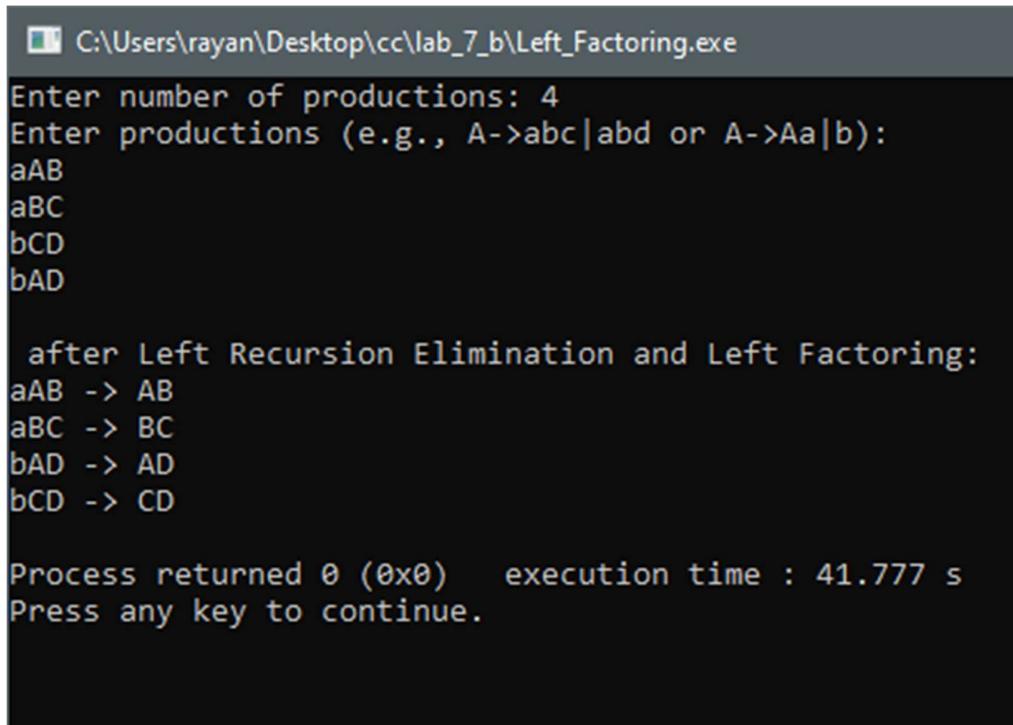
aAB

aBC

bCD

bAD

Output:



```
C:\Users\rayan\Desktop\cc\lab_7_b\Left_Factoring.exe
Enter number of productions: 4
Enter productions (e.g., A->abc|abd or A->Aa|b):
aAB
aBC
bCD
bAD

after Left Recursion Elimination and Left Factoring:
aAB -> AB
aBC -> BC
bAD -> AD
bCD -> CD

Process returned 0 (0x0)  execution time : 41.777 s
Press any key to continue.
```

Figure 1: Output showing factored grammar after left factoring.

Discussion:

Initially, I faced difficulty in detecting the longest common prefix among multiple productions. I implemented the function `findLongestPrefix()` which compares all pairs of productions, finds the longest common prefix, and counts how many productions share it.

Another challenge was handling epsilon (ϵ) in the factored output. When a production is exactly equal to the common prefix, the remaining part is empty and I needed to represent it as epsilon in the new non-terminal. I solved this by checking the length of the remaining string after removing the prefix and explicitly inserting epsilon when the string was empty.