Lab Report: Eliminating Left Recursion and Left Factoring in a Grammar

Objective

The objective of this lab is to understand and implement programs to:

1. Eliminate left recursion in a grammar.
2. Perform left factoring on a grammar to make it suitable for top-down parsing.

Problem Statements

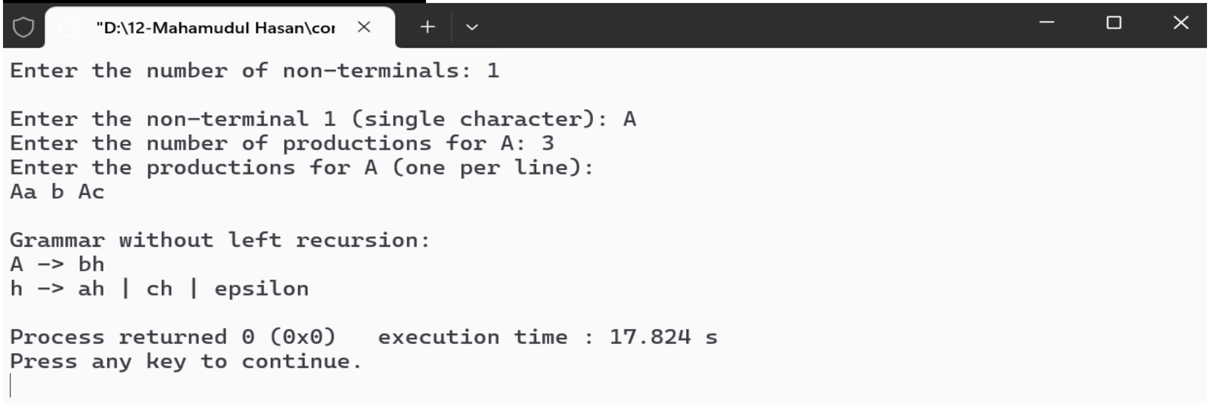
1. **Problem 1:**
   Write a program to eliminate the left recursion from the given grammar using C programming.
2. **Problem 2:**
   Write a program to perform left factoring on the given grammar to remove ambiguities in parsing.

Program Code

Program 1: Removing Left Recursion

```
#include <stdio.h>
#include <string.h>
#define MAX_RULES 10
#define MAX_LENGTH 100
#define MAX_NON_TERMINALS 10

typedef struct {
    char non_terminal;
    char alpha[MAX_RULES][MAX_LENGTH];
    char beta[MAX_RULES][MAX_LENGTH];
    int alpha_count;
    int beta_count;
} GrammarRule;

void remove_left_recursion(GrammarRule* rule) {
    if (rule->alpha_count == 0) {
        printf("%c -> ", rule->non_terminal);
        for (int i = 0; i < rule->beta_count; i++) {
            printf("%s", rule->beta[i]);
            if (i < rule->beta_count - 1) printf(" | ");
        }
        printf("\n");
        return;
    }
    char new_non_terminal = rule->non_terminal + '\'';
    printf("%c -> ", rule->non_terminal);
    for (int i = 0; i < rule->beta_count; i++) {
        printf("%s%c", rule->beta[i], new_non_terminal);
        if (i < rule->beta_count - 1) printf(" | ");
    }
    printf("\n");
    printf("%c -> ", new_non_terminal);
    for (int i = 0; i < rule->alpha_count; i++) {
        printf("%s%c", rule->alpha[i], new_non_terminal);
        if (i < rule->alpha_count - 1) printf(" | ");
    }
    printf(" | epsilon\n");
}

int main() {
    GrammarRule rules[MAX_NON_TERMINALS];
    int num_non_terminals;

    printf("Enter the number of non-terminals: ");
    scanf("%d", &num_non_terminals);

    for (int r = 0; r < num_non_terminals; r++) {
        GrammarRule* rule = &rules[r];
        rule->alpha_count = 0;
        rule->beta_count = 0;
        printf("\nEnter the non-terminal %d (single character): ", r + 1);
        scanf(" %c", &rule->non_terminal);
        int n;
        printf("Enter the number of productions for %c: ", rule-
>non_terminal);
        scanf("%d", &n);
        printf("Enter the productions for %c (one per line):\n", rule-
>non_terminal);
        for (int i = 0; i < n; i++) {
            char production[MAX_LENGTH];
            scanf("%s", production);
            if (production[0] == rule->non_terminal) {
                strcpy(rule->alpha[rule->alpha_count++], production + 1);
            } else {
                strcpy(rule->beta[rule->beta_count++], production);
            }
        }
    }

    printf("\nGrammar without left recursion:\n");
    for (int r = 0; r < num_non_terminals; r++) {
        remove_left_recursion(&rules[r]);
    }

    return 0;
}
```

Sample Input & Output:

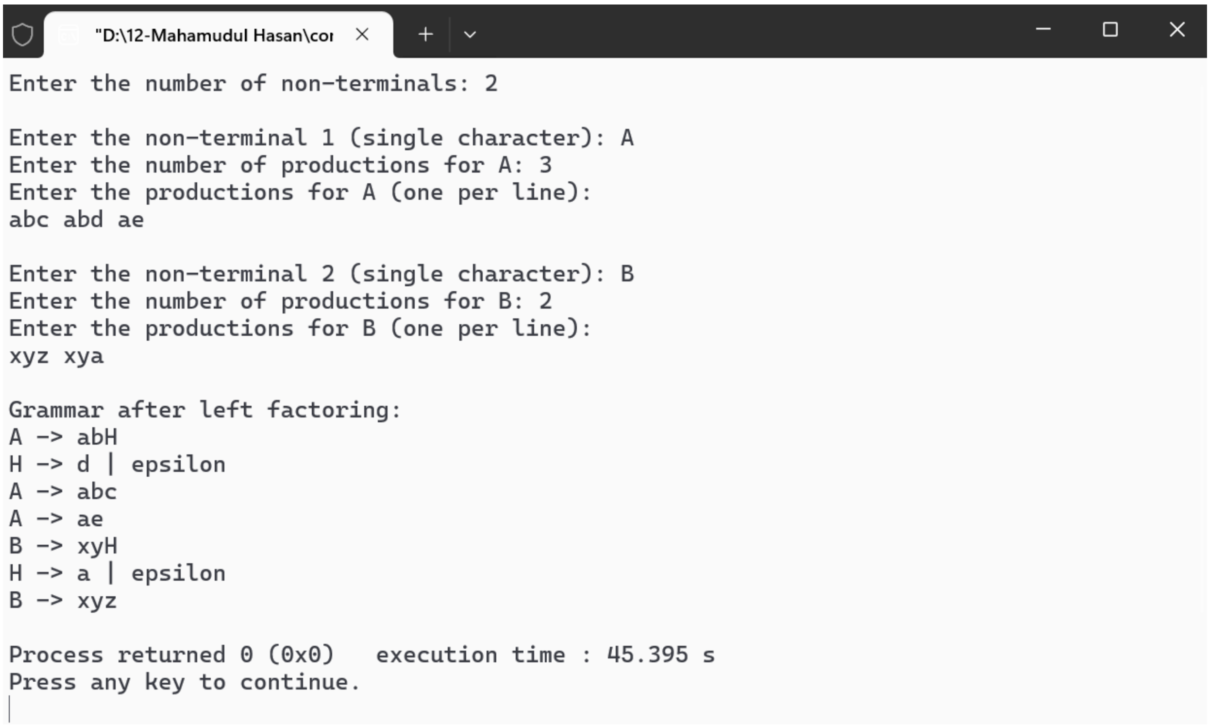


```
Enter the number of non-terminals: 1

Enter the non-terminal 1 (single character): A
Enter the number of productions for A: 3
Enter the productions for A (one per line):
Aa b Ac

Grammar without left recursion:
A -> bh
h -> ah | ch | epsilon

Process returned 0 (0x0)   execution time : 17.824 s
Press any key to continue.
```

Program 2: Performing Left Factoring

```
#include <stdio.h>
#include <string.h>
#define MAX_RULES 10
#define MAX_LENGTH 100
#define MAX_NON_TERMINALS 10

typedef struct {
    char non_terminal;
    char productions[MAX_RULES][MAX_LENGTH];
    int production_count;
} GrammarRule;

int longest_common_prefix(char *str1, char *str2) {
    int i = 0;
    while (str1[i] && str2[i] && str1[i] == str2[i]) {
        i++;
    }
    return i;
}

void left_factoring(GrammarRule* rule) {
    int prefix_length;
    int common_prefix[MAX_RULES] = {0};
    for (int i = 0; i < rule->production_count; i++) {
        if (common_prefix[i]) continue;
        for (int j = i + 1; j < rule->production_count; j++) {
            prefix_length = longest_common_prefix(rule->productions[i],
rule->productions[j]);
            if (prefix_length > 0) {
                char prefix[MAX_LENGTH];
                strncpy(prefix, rule->productions[i], prefix_length);
                prefix[prefix_length] = '\0';
                printf("%c -> %sH\n", rule->non_terminal, prefix);
                printf("H -> ");
                int first = 1;
                for (int k = i; k < rule->production_count; k++) {
                    if (longest_common_prefix(rule->productions[i], rule-
>productions[k]) == prefix_length) {
                        if (!first) printf(" | ");
                        printf("%s", rule->productions[k] + prefix_length);
                        common_prefix[k] = 1;
                        first = 0;
                    }
                }
                printf(" | epsilon\n");
                break;
            }
        }
    }
    for (int i = 0; i < rule->production_count; i++) {
        if (!common_prefix[i]) {
            printf("%c -> %s\n", rule->non_terminal, rule->productions[i]);
        }
    }
}

int main() {
    GrammarRule rules[MAX_NON_TERMINALS];
    int num_non_terminals;

    printf("Enter the number of non-terminals: ");
    scanf("%d", &num_non_terminals);

    for (int r = 0; r < num_non_terminals; r++) {
        GrammarRule* rule = &rules[r];
        rule->production_count = 0;
        printf("\nEnter the non-terminal %d (single character): ", r + 1);
        scanf(" %c", &rule->non_terminal);
        int n;
        printf("Enter the number of productions for %c: ", rule-
>non_terminal);
        scanf("%d", &n);
        printf("Enter the productions for %c (one per line):\n", rule-
>non_terminal);
        for (int i = 0; i < n; i++) {
            scanf("%s", rule->productions[rule->production_count++]);
        }
    }

    printf("\nGrammar after left factoring:\n");
    for (int r = 0; r < num_non_terminals; r++) {
        left_factoring(&rules[r]);
    }

    return 0;
}
```

Sample Input and Output



```
Enter the number of non-terminals: 2

Enter the non-terminal 1 (single character): A
Enter the number of productions for A: 3
Enter the productions for A (one per line):
abc abd ae

Enter the non-terminal 2 (single character): B
Enter the number of productions for B: 2
Enter the productions for B (one per line):
xyz xya

Grammar after left factoring:
A -> abH
H -> d | epsilon
A -> abc
A -> ae
B -> xyH
H -> a | epsilon
B -> xyz

Process returned 0 (0x0)   execution time : 45.395 s
Press any key to continue.
```

Conclusion

The programs successfully eliminate left recursion and perform left factoring, enabling top-down parsing for grammars. The structured approach to modifying the grammar ensures compatibility with LL(1) parsers.