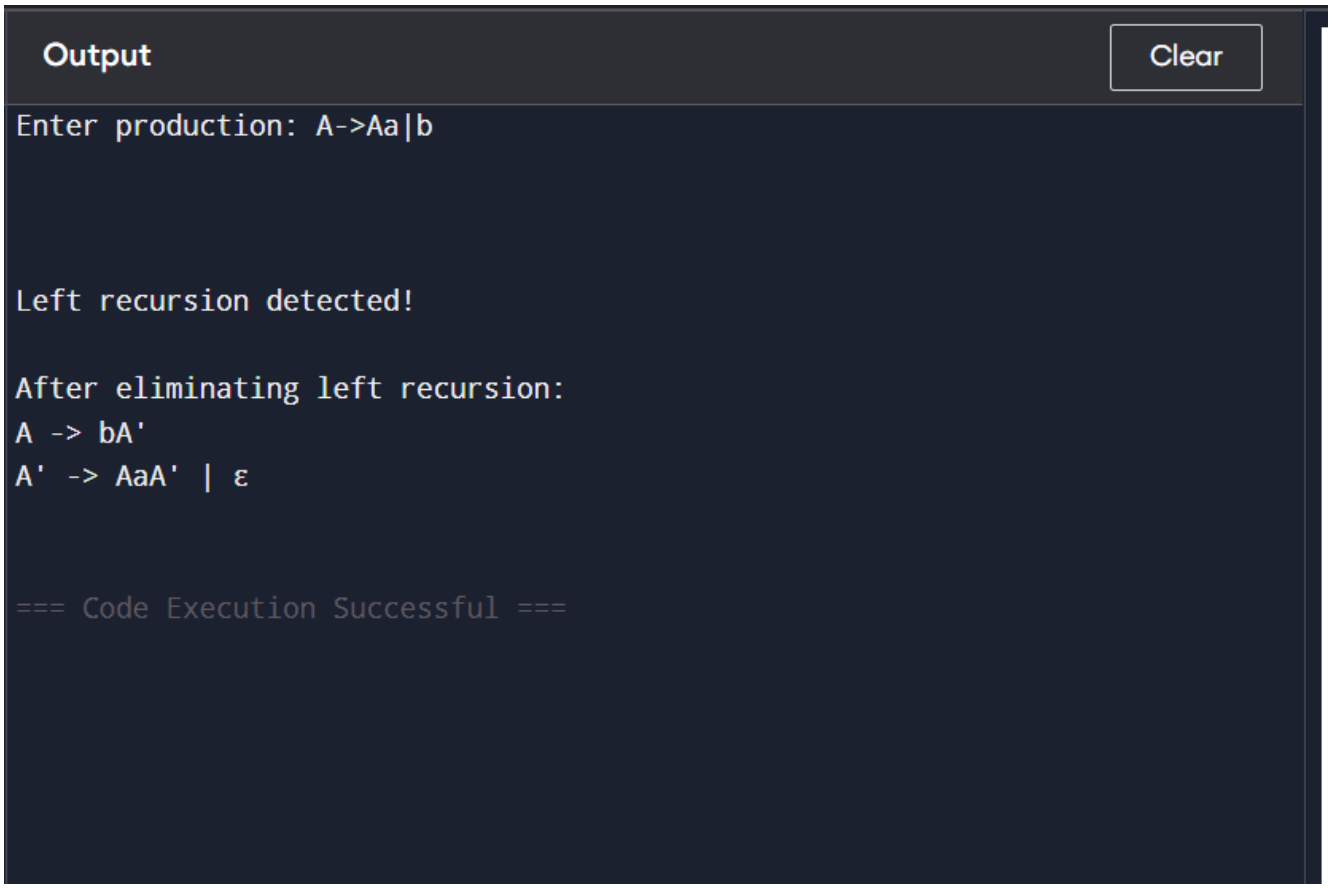
Experiment 6:

Implement a C program to eliminate left recursion.

Code:

```
#include <stdio.h>
#include <string.h>
int main() {
   char input[100], nonTerminal, beta[10], alpha[10];
   int i = 3, j = 0;
   printf("Enter production: ");
   scanf("%s", input);
   nonTerminal = input[0];
   if (input[0] == input[3]) {
      printf("\nLeft recursion detected!\n");
      while (input[i] != '|' && input[i] != '\0')
         alpha[j++] = input[i++];
      alpha[j] = '\0';
      j = 0;
      i++;
      while (input[i] != '\0')
         beta[j++] = input[i++];
      beta[j] = '\0';
      printf("\nAfter eliminating left recursion:\n");
      printf("%c -> %s%c'\n", nonTerminal, beta, nonTerminal);
      printf("%c' -> %s%c' | ε\n", nonTerminal, alpha, nonTerminal);
   }
   else {
      printf("\nNo left recursion detected.\n");
   }
   return 0;
}
```

Output:


Output
Clear
Enter production: A->Aa|b
Left recursion detected!
After eliminating left recursion:
A -> bA'
A' -> AaA' | ε
=== Code Execution Successful ===

Experiment 7:

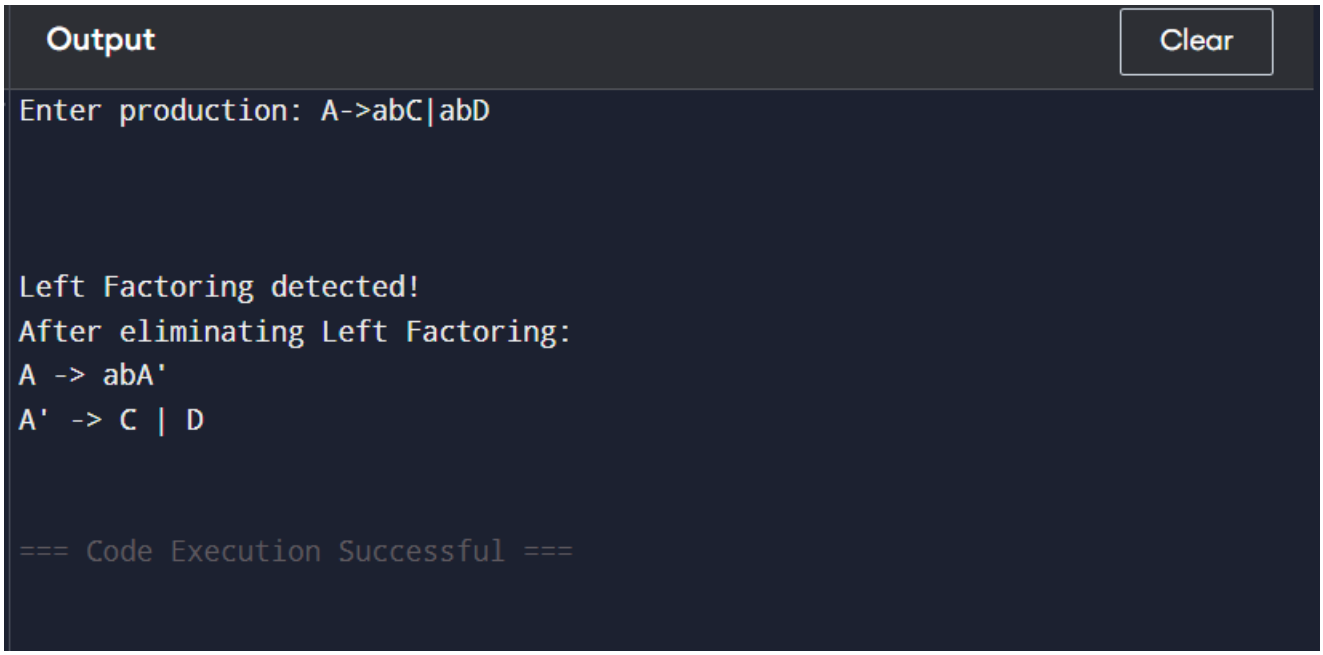
Implement a C program to eliminate left factoring.

Code:

```
#include <stdio.h>
#include <string.h>
int main() {
    char input[100], common[10], alpha[10], beta[10];
    int i = 0, j = 0, k = 0, pos;
    printf("Enter production: ");
    scanf("%s", input);
    for (i = 0; input[i] != '|'; i++);
    pos = i;
    char left[20], right1[20], right2[20];
    strncpy(left, input, 1);
    left[1] = '\0';
    strncpy(right1, input + 3, pos - 3);
    right1[pos - 3] = '\0';
    strcpy(right2, input + pos + 1);
    i = 0;
    while (right1[i] == right2[i] && right1[i] != '\0') {
        common[k++] = right1[i];
        i++;
    }
    common[k] = '\0';
    strcpy(alpha, right1 + i);
    strcpy(beta, right2 + i);
    if (strlen(common) == 0) {
        printf("\nNo Left Factoring found.\n");
    } else {
        printf("\nLeft Factoring detected!\n");
        printf("After eliminating Left Factoring:\n");
        printf("%s -> %s%s'\n", left, common, left);
        printf("%s' -> %s | %s\n", left, alpha[0] ? alpha : "ε", beta[0] ? beta : "ε");
    }
```

```
    return 0;
}
```

Output:



Experiment 8:

Implement a C program to perform symbol table operations.

Code:

```
#include <stdio.h>
#include <string.h>
struct Symbol {
    char name[20];
    char type[10];
    int value;
};
int main() {
    struct Symbol table[50];
    int n, i;
    printf("Enter number of symbols: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("\nEnter symbol %d name: ", i+1);
        scanf("%s", table[i].name);
        printf("Enter symbol %d type: ", i+1);
        scanf("%s", table[i].type);
        printf("Enter symbol %d value: ", i+1);
        scanf("%d", &table[i].value);
    }
    printf("\nSymbol Table:\n");
    printf("Name\tType\tValue\n");
    printf("----------------------\n");
    for (i = 0; i < n; i++) {
        printf("%s\t%s\t%d\n", table[i].name, table[i].type, table[i].value);
    }
    return 0;
}
```

Output:

Output

Clear

```
Enter number of symbols: 2

Enter symbol 1 name: y
Enter symbol 1 type: int
Enter symbol 1 value: 45

Enter symbol 2 name: x
Enter symbol 2 type: int
Enter symbol 2 value: 2

Symbol Table:
Name    Type    Value
----------------------
y   int 45
x   int 2


=== Code Execution Successful ===
```