RemoveLeftRecursion

removeLeftRecr.c > ...

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

#define MAX 100

// Production structure
typedef struct {
    char left[MAX];
    char right[MAX][MAX];
    int count;
} Production;

// Eliminate Left Recursion
void eliminate_left_recursion(Production *prod) {
    char new_non_terminal[MAX];
    sprintf(new_non_terminal, "%c'", prod->left[0]);

    printf("%s -> ", prod->left);
    for (int i = 0; i < prod->count; i++) {
        if (prod->right[i][0] == prod->left[0]) {
            printf("%s' ", prod->left);
            break;
        } else {
            printf("%s ", prod->right[i]);
        }
    }
    printf("\n");

    printf("%s' -> ", new_non_terminal);
    for (int i = 0; i < prod->count; i++) {
        if (prod->right[i][0] == prod->left[0]) {
            printf("%s ", prod->right[i] + 1);
            printf("%s' ", new_non_terminal);
        }
    }
    printf("| ε\n");
}

// Main function to process grammar
int main() {
    int num_rules;
    printf("Enter number of production rules: ");
```

```c

// Main function to process grammar
int main() {
    int num_rules;
    printf("Enter number of production rules: ");
    scanf("%d", &num_rules);
    getchar();

    for (int i = 0; i < num_rules; i++) {
        Production prod;
        printf("\nEnter left side (e.g., A): ");
        scanf("%s", prod.left);
        getchar();

        printf("Enter number of right-hand side productions: ");
        scanf("%d", &prod.count);
        getchar();

        printf("Enter productions:\n");
        for (int j = 0; j < prod.count; j++) {
            printf("Production %d: ", j + 1);
            scanf("%s", prod.right[j]);
        }

        eliminate_left_recursion(&prod);
    }

    return 0;
}


//OUTPUT

// Enter number of production rules: 1
// Enter left side (e.g., A): E
// Enter number of right-hand side productions: 2
// Enter productions:
// Production 1: E+T
// Production 2: T

```

Syntaxnalyzer

```
41
42  // Main function to drive the parser
43  int main() {
44      // Read the input expression
45      printf("Enter an arithmetic expression: ");
46      fgets(input, MAX_INPUT, stdin);
47      input[strcspn(input, "\n")] = '\0'; // Remove newline
48
49      // Start parsing
50      parse_expr();
51      printf("The expression is syntactically correct.\n");
52      return 0;
53  }
54
55  // Get the next token from the input
56  Token get_next_token() {
57      Token token;
58
59      // Skip white spaces
60      while (input[pos] != '\0' && isspace(input[pos])) {
61          pos++;
62      }
63
64      // End of input
65      if (input[pos] == '\0') {
66          token.type = END;
67          return token;
68      }
69
70      // Numbers
71      if (isdigit(input[pos])) {
72          token.type = NUM;
73          token.value = 0;
74          while (isdigit(input[pos])) {
75              token.value = token.value * 10 + (input[pos] - '0');
76              pos++;
77          }
78          return token;
79      }
80
81      // Operators and parentheses
```

```
80
81      // Operators and parentheses
82      if (input[pos] == '+') {
83          token.type = PLUS;
84          pos++;
85          return token;
86      }
87      if (input[pos] == '*') {
88          token.type = STAR;
89          pos++;
90          return token;
91      }
92      if (input[pos] == '(') {
93          token.type = LPAREN;
94          pos++;
95          return token;
96      }
97      if (input[pos] == ')') {
98          token.type = RPAREN;
99          pos++;
00          return token;
01      }
02
03      // Invalid character
04      token.type = INVALID;
05      return token;
06  }
07
08  // Match the expected token type, otherwise throw error
09  void match(TokenType expected) {
10      Token token = get_next_token();
11      if (token.type != expected) {
12          error("Unexpected token.");
13      }
14  }
15
16  // Parse an expression: expr -> term + expr | term
17  void parse_expr() {
18      parse_term();
19
```

```
}

// Match the expected token type, otherwise throw error
void match(TokenType expected) {
    Token token = get_next_token();
    if (token.type != expected) {
        error("Unexpected token.");
    }
}

// Parse an expression: expr -> term + expr | term
void parse_expr() {
    parse_term();

    Token token = get_next_token();
    while (token.type == PLUS) {
        match(PLUS);
        parse_term();
        token = get_next_token();
    }
}

// Parse a term: term -> factor * term | factor
void parse_term() {
    parse_factor();

    Token token = get_next_token();
    while (token.type == STAR) {
        match(STAR);
        parse_factor();
        token = get_next_token();
    }
}

// Parse a factor: factor -> ( expr ) | num
void parse_factor() {
    Token token = get_next_token();

    if (token.type == NUM) {
        return; // Valid number
    } else if (token.type == LPAREN) {
```

```
// Parse a term: term -> factor * term | factor
void parse_term() {
    parse_factor();

    Token token = get_next_token();
    while (token.type == STAR) {
        match(STAR);
        parse_factor();
        token = get_next_token();
    }
}

// Parse a factor: factor -> ( expr ) | num
void parse_factor() {
    Token token = get_next_token();

    if (token.type == NUM) {
        return; // Valid number
    } else if (token.type == LPAREN) {
        parse_expr();
        match(RPAREN); // Expect closing parenthesis
    } else {
        error("Invalid factor.");
    }
}


//Enter an arithmetic expression: 3 + 5 * (2 + 8)
//The expression is syntactically correct.
```

SymbolTable

symbolTable.c > ...

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

#define SIZE 100  // Max number of symbols

// Structure to hold individual symbol data
struct Symbol {
    char name[30];
    char type[30];
    int size;
    int line;
};

// Symbol Table structure
struct SymbolTable {
    struct Symbol symbols[SIZE];
    int count;
};

// Insert a new symbol into the table
void insert(struct SymbolTable* table, char name[], char type[], int size, int line) {
    // Check if the symbol already exists
    for (int i = 0; i < table->count; i++) {
        if (strcmp(table->symbols[i].name, name) == 0) {
            printf("Symbol '%s' already exists in the table.\n", name);
            return;
        }
    }

    // Add the new symbol
    strcpy(table->symbols[table->count].name, name);
    strcpy(table->symbols[table->count].type, type);
    table->symbols[table->count].size = size;
    table->symbols[table->count].line = line;
    table->count++;

    printf("Symbol '%s' inserted successfully.\n", name);
}

// Search for a symbol by name and return its index
int search(struct SymbolTable* table, char name[]) {
```

```c
}

// Search for a symbol by name and return its index
int search(struct SymbolTable* table, char name[]) {
    for (int i = 0; i < table->count; i++) {
        if (strcmp(table->symbols[i].name, name) == 0)
            return i;
    }
    return -1;
}

// Display the entire symbol table
void display(struct SymbolTable* table) {
    printf("\n%-10s %-10s %-5s %-5s\n", "Name", "Type", "Size", "Line");
    printf("-----------------------------------\n");
    for (int i = 0; i < table->count; i++) {
        printf("%-10s %-10s %-5d %-5d\n", table->symbols[i].name,
               table->symbols[i].type, table->symbols[i].size,
               table->symbols[i].line);
    }
}

// Main function to demonstrate usage
int main() {
    struct SymbolTable table;
    table.count = 0;

    // Inserting symbols
    insert(&table, "x", "int", 4, 1);
    insert(&table, "y", "float", 4, 2);
    insert(&table, "z", "char", 1, 3);

    // Searching for a symbol
    printf("\nSearching for 'y':\n");
    int index = search(&table, "y");
    if (index != -1)
        printf("Symbol found at index %d.\n", index);
    else
        printf("Symbol not found.\n");

    // Display the table
```

```c
int main() {
    struct SymbolTable table;
    table.count = 0;

    // Inserting symbols
    insert(&table, "x", "int", 4, 1);
    insert(&table, "y", "float", 4, 2);
    insert(&table, "z", "char", 1, 3);

    // Searching for a symbol
    printf("\nSearching for 'y':\n");
    int index = search(&table, "y");
    if (index != -1)
        printf("Symbol found at index %d.\n", index);
    else
        printf("Symbol not found.\n");

    // Display the table
    printf("\nSymbol Table:\n");
    display(&table);

    return 0;
}

// Symbol Table:

// Name       Type       Size  Line
// ---------------------------------
// x          int        4     1
// y          float      4     2
// z          char       1     3
// PS C:\shahin\c programms>
```