IUBAT— International University of Business Agriculture and Technology

Founded 1991 by Md. Alimullah Miyan

Final Assignment

Course Name: Programming Languages and Structures

Course Code: CSC 461

Submitted To

Md. Nazir Ahmed

Lecturer

CSE Dept. IUBAT

Submitted by

Rakibul Hasan

ID: 21203070

Submission Date: January 20, 2025

Title: Recursive Descent Parser Implementation in C

Objective:
The purpose of this assignment was to design and implement a Recursive Descent Parser in the C programming language. The parser validates and evaluates input mathematical expressions according to the given grammar.

Grammar Specification

The following grammar, written in Backus-Naur Form (BNF), was used:

1. <expression> => <term> { ("+" | "-") <term> }
2. <term> => <factor> { ("*" | "/") <factor> }
3. <factor> => <number> | "(" <expression> ")"
4. <number> => <digit> { <digit> }
5. <digit> => "0" | "1" | ... | "9"

Assignment Requirements

1. **Lexer Implementation**: Tokenizes the input into numbers, operators, and parentheses.
2. **Parser Implementation**: Validates expressions and parses them according to the grammar.
3. **Evaluation**: Calculates the result for valid expressions.
4. **Error Handling**: Identifies and displays error messages for invalid inputs.
5. **Testing**: Includes test cases for both valid and invalid expressions.

Code Explanation

Global Variables and Token Types

- **TokenType**: Enum representing different token types (e.g., NUMBER, PLUS, etc.).
- **Token Structure**: Contains the type and value of a token.
- **Global Variables**:
  - input: Pointer to the input string.
  - currentToken: Stores the currently processed token.

Function Descriptions

1. **getNextToken()**:
    - Tokenizes the input string.
    - Skips spaces, identifies numbers, operators, and parentheses.
2. **evaluateExpression()**:
    - Parses and evaluates expressions with + and - operators.
3. **evaluateTerm()**:
    - Parses and evaluates terms with * and / operators.
    - Handles division by zero.
4. **evaluateFactor()**:
    - Parses numbers and handles nested expressions within parentheses.
5. **throwError(const char *message)**:
    - Displays error messages and exits the program.
6. **main()**:
    - Reads input, tokenizes, validates, and evaluates expressions.
    - Prints results for valid expressions or error messages for invalid inputs.

Code

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

// Define Token Types
typedef enum {
   NUMBER, PLUS, MINUS, MULTIPLY, DIVIDE, LPAREN, RPAREN, END,
INVALID
} TokenType;

// Define Token Structure
typedef struct {
   TokenType type;
   int value;
} Token;
```

```c
char *input;
Token currentToken;

void getNextToken();
int evaluateExpression();
int evaluateTerm();
int evaluateFactor();
void throwError(const char *message);

// Function to Fetch the Next Token
void getNextToken() {
    // Skip spaces
    while (isspace(*input)) input++;

    if (*input == '\0') { // End of input
        currentToken.type = END;
        return;
    }

    if (isdigit(*input)) { // Handle numbers
        currentToken.type = NUMBER;
        currentToken.value = 0;
        while (isdigit(*input)) {
            currentToken.value = currentToken.value * 10 + (*input - '0');
            input++;
        }
        return;
    }

    // Handle operators and parentheses
    switch (*input) {
        case '+': currentToken.type = PLUS; input++; break;
        case '-': currentToken.type = MINUS; input++; break;
        case '*': currentToken.type = MULTIPLY; input++; break;
        case '/': currentToken.type = DIVIDE; input++; break;
        case '(': currentToken.type = LPAREN; input++; break;
        case ')': currentToken.type = RPAREN; input++; break;
        default: currentToken.type = INVALID; input++; break;
    }
}

// Throw Error and Exit
void throwError(const char *message) {
    printf("Error: %s\n", message);
    exit(EXIT_FAILURE);
}

// Parse and Evaluate an Expression (Handles +, -)
int evaluateExpression() {
    int result = evaluateTerm();

    while (currentToken.type == PLUS || currentToken.type == MINUS) {
        TokenType operator = currentToken.type;
        getNextToken();
        int nextValue = evaluateTerm();

        if (operator == PLUS) {
            result += nextValue;
        } else {
            result -= nextValue;
        }
    }

    return result;
}

// Parse and Evaluate a Term (Handles *, /)
int evaluateTerm() {
    int result = evaluateFactor();

    while (currentToken.type == MULTIPLY || currentToken.type == DIVIDE) {
        TokenType operator = currentToken.type;
        getNextToken();
        int nextValue = evaluateFactor();

        if (operator == MULTIPLY) {
            result *= nextValue;
        } else {
            if (nextValue == 0) {
                throwError("Division by zero is not allowed.");
            }
            result /= nextValue;
        }
    }

    return result;
}

// Parse and Evaluate a Factor (Handles numbers and parentheses)
int evaluateFactor() {
    if (currentToken.type == NUMBER) {
        int value = currentToken.value;
        getNextToken();
        return value;
    } else if (currentToken.type == LPAREN) {
        getNextToken();
        int value = evaluateExpression();
        if (currentToken.type != RPAREN) {
            throwError("Missing closing parenthesis.");
        }
        getNextToken();
        return value;
    } else {
        throwError("Invalid input.");
        return 0; // Unreachable
    }
}
```

```c
// Main Function
int main() {
    char inputBuffer[256];

    printf("Enter an expression: ");
    if (!fgets(inputBuffer, sizeof(inputBuffer), stdin)) {
        throwError("Failed to read input.");
    }

    input = inputBuffer;
    getNextToken();

    int result = evaluateExpression();

    if (currentToken.type != END) {
        throwError("Unexpected input at the end.");
    }
    if(result){
        printf("Valid Expression\n");
        printf("Result: %d\n", result);
    }
    return 0;
}
```

Test Cases

Valid Inputs

Input	**Output**
1 + 2	Valid Expression, Result: 3
5 * (8 + 1)	Valid Expression, Result: 45
15 / 3 + 6	Valid Expression, Result: 11

Invalid Inputs

Input	Error Message
9 +	Error: Unexpected input at the end.
(1 / 5	Error: Missing closing parenthesis.
1 / 0	Error: Division by zero is not allowed.