**EX 10:** **Write a C program to construct recursive descent parsing.**

**Aim:**

To implement a Recursive Descent Parser in C to check whether an input string follows a predefined grammar.

**Algorithm:**

1. **Define a Grammar**:
   * Use recursive functions to represent the grammar rules.
2. **Read Input String**:
   * Accept an arithmetic expression as input.
3. **Parse the String Using Recursive Functions**:
   * Implement functions for each non-terminal (E, T, F).
4. **Check If Parsing Is Successful**:
   * If all characters are consumed and no syntax errors occur, the string is valid.
   * Otherwise, it is invalid.

**CODE:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

char input[100]; // Input string

int pos = 0; // Current position in input

// Function prototypes

void E();

void T();

void F();

// Function to match a character and move to the next

void match(char expected) {

if (input[pos] == expected)

pos++;

else {

printf("Error: Unexpected character '%c'\n", input[pos]);

exit(1);

}

}

// Recursive function for E → T + E | T

void E() {

T(); // Parse T

if (input[pos] == '+') { // Check for +

match('+');

E(); // Parse E recursively

}

}

// Recursive function for T → F \* T | F

void T() {

F(); // Parse F

if (input[pos] == '\*') { // Check for \*

match('\*');

T(); // Parse T recursively

}

}

// Recursive function for F → (E) | id

void F() {

if (isalpha(input[pos])) { // If it's an identifier (id)

match(input[pos]);

} else if (input[pos] == '(') { // If it's a left parenthesis

match('(');

E(); // Parse E inside parentheses

match(')'); // Match closing parenthesis

} else {

printf("Error: Invalid character '%c'\n", input[pos]);

exit(1);

}

}

// Main function

int main() {

printf("Enter an expression: ");

scanf("%s", input);

E(); // Start parsing from E

if (input[pos] == '\0') // If the entire input is consumed

printf("Valid Expression!\n");

else

printf("Error: Unexpected characters after parsing!\n");

return 0;

}

