# Program 15

Write a C Program to implement the operator precedence parsing.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100

char stack[MAX];

int top = -1;

char input[MAX];

// Operator precedence table

int precedence(char op) {

   if (op == '\*' || op == '/') return 2;

   if (op == '+' || op == '-') return 1;

   return 0; // Default for non-operators

}

// Push onto stack

void push(char symbol) {

   stack[++top] = symbol;

}

// Pop from stack

char pop() {

   return (top >= 0) ? stack[top--] : '\0';

}

// Get the top of the stack

char peek() {

   return (top >= 0) ? stack[top] : '$'; // '$' is used as a bottom marker

}

// Check if character is an operator

int is\_operator(char ch) {

   return (ch == '+' || ch == '-' || ch == '\*' || ch == '/');

}

// Perform reduction when possible

void reduce() {

   while (top >= 2) {

       // Check if stack contains "E op E" pattern

       if (stack[top] == 'E' && is\_operator(stack[top - 1]) && stack[top - 2] == 'E') {

           char op = stack[top - 1]; // Store the operator

           pop(); // Remove 'E'

           pop(); // Remove operator

           pop(); // Remove 'E'

           push('E'); // Replace with single 'E'

           printf("Reduce: E %c E → E\n", op);

       } else {

           break; // Stop reducing if no match

       }

   }

}

// Operator Precedence Parsing Algorithm

void operator\_precedence\_parsing() {

   int position = 0;

   push('$'); // Push bottom marker

   printf("\nOperator Precedence Parsing Steps:\n");

   while (position < strlen(input)) {

       char current = input[position];

       if (strncmp(&input[position], "id", 2) == 0) { // If "id" is encountered

           printf("Shift: id\n");

           push('E'); // Reduce "id" → E immediately

           position += 2; // Move past "id"

       }

       else if (is\_operator(current)) { // If operator is encountered

           while (is\_operator(peek()) && precedence(peek()) >= precedence(current)) {

               reduce(); // Reduce before shifting new operator

           }

           printf("Shift: %c\n", current);

           push(current);

           position++;

       }

       else {

           printf("Invalid character detected: %c\n", current);

           exit(1);

       }

   }

   // Final reduction to ensure only 'E' remains

   reduce();

   if (top == 1 && stack[top] == 'E' && stack[0] == '$') {

       printf("\nParsing successful!\n");

   } else {

       printf("\nParsing failed!\n");

   }

}

int main() {

   printf("Enter an arithmetic expression: ");

   scanf("%s", input);

   operator\_precedence\_parsing();

   return 0;

}

