Evaluating Arithmetic Expression

Write a C program to evaluate Arithmetic expression using stack.

PROGRAM:

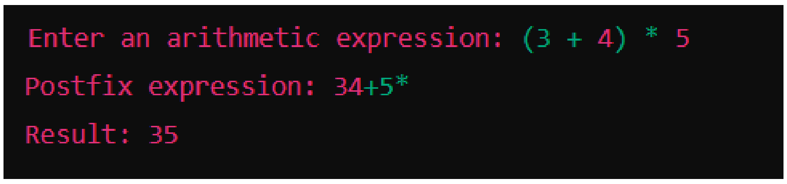
```c
#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>


#define MAX 100


// Stack structure for characters (operators and parentheses)

struct CharStack {

   char arr[MAX];

   int top;

};


// Stack structure for integers (operands)

struct IntStack {

   int arr[MAX];

   int top;

};


// Initialize the character stack

void initCharStack(struct CharStack* stack) {

   stack->top = -1;

}


// Initialize the integer stack

void initIntStack(struct IntStack* stack) {

   stack->top = -1;

}


// Check if the character stack is empty
```

```
int isCharStackEmpty(struct CharStack* stack) {
    return stack->top == -1;
}

// Check if the integer stack is empty
int isIntStackEmpty(struct IntStack* stack) {
    return stack->top == -1;
}

// Push a character onto the character stack
void charPush(struct CharStack* stack, char op) {
    stack->arr[++stack->top] = op;
}

// Push an integer onto the integer stack
void intPush(struct IntStack* stack, int num) {
    stack->arr[++stack->top] = num;
}

// Pop a character from the character stack
char charPop(struct CharStack* stack) {
    if (isCharStackEmpty(stack)) {
        return '\0';
    }
    return stack->arr[stack->top--];
}

// Pop an integer from the integer stack
int intPop(struct IntStack* stack) {
    if (isIntStackEmpty(stack)) {
        return -1;
    }
    return stack->arr[stack->top--];
}
```

```
// Peek the top character from the character stack
char charPeek(struct CharStack* stack) {
    if (isCharStackEmpty(stack)) {
        return '\0';
    }
    return stack->arr[stack->top];
}

// Check if a character is an operator
int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

// Check the precedence of operators
int precedence(char op) {
    if (op == '+' || op == '-') {
        return 1;
    } else if (op == '*' || op == '/') {
        return 2;
    }
    return 0;
}

// Convert infix expression to postfix expression
void infixToPostfix(char* infix, char* postfix) {
    struct CharStack stack;
    initCharStack(&stack);
    int k = 0;

    for (int i = 0; infix[i] != '\0'; i++) {
        if (isdigit(infix[i])) {
            postfix[k++] = infix[i];
        } else if (infix[i] == '(') {
            charPush(&stack, infix[i]);
        } else if (infix[i] == ')') {
            while (!isCharStackEmpty(&stack) && charPeek(&stack) != '(') {
                postfix[k++] = charPop(&stack);
            }
            charPop(&stack); // Pop the '('
        } else if (isOperator(infix[i])) {
            while (!isCharStackEmpty(&stack) && precedence(charPeek(&stack)) >= precedence(infix[i])) {
                postfix[k++] = charPop(&stack);
            }
            charPush(&stack, infix[i]);
        }
    }

    while (!isCharStackEmpty(&stack)) {
        postfix[k++] = charPop(&stack);
    }

    postfix[k] = '\0';
}

// Evaluate the postfix expression
int evaluatePostfix(char* postfix) {
    struct IntStack stack;
    initIntStack(&stack);

    for (int i = 0; postfix[i] != '\0'; i++) {
        if (isdigit(postfix[i])) {
            intPush(&stack, postfix[i] - '0');
        } else if (isOperator(postfix[i])) {
            int val2 = intPop(&stack);
            int val1 = intPop(&stack);

            switch (postfix[i]) {
```

```
            case '+': intPush(&stack, val1 + val2); break;
            case '-': intPush(&stack, val1 - val2); break;
            case '*': intPush(&stack, val1 * val2); break;
            case '/': intPush(&stack, val1 / val2); break;
        }
      }
    }

    return intPop(&stack);
}

int main() {
    char infix[MAX], postfix[MAX];

    printf("Enter an arithmetic expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);
    printf("Result: %d\n", evaluatePostfix(postfix));

    return 0;
}
```

OUTPUT:



```
Enter an arithmetic expression: (3 + 4) * 5
Postfix expression: 34+5*
Result: 35
```