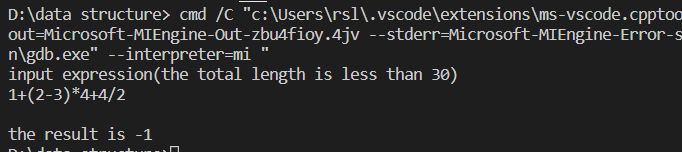
测试样例

1+(2-3)\*4+4/2

测试结果



算法和数据结构设计

1. 建立存储数字和运算符的栈
2. 建立输入多项式，数字和运算符压栈出栈，判断运算符级别，运算的函数
3. 输入多项式，初始化存运算符和数字的栈
4. 若某一项是数字，进入存数字的栈，若是运算符，判断运算符级别，若运算符级别低，弹出前一个运算符，和两个数字运算，结果和最新的运算符再次压栈。以此进行下去。
5. 读取多项式完成后，弹出两个数字，一个运算符，运算后结果进入数字栈，依次进行，直到运算符栈为空。

#include <stdio.h>

#include <stdlib.h>

#define STACK\_LEN 30  //定义栈的长度

//存储数字的栈

typedef struct stack\_num {

    int top;

    int sta[STACK\_LEN];

}Stack\_num;

//存储运算符的栈

typedef struct stack\_op {

    int top;

    char sta[STACK\_LEN];

}Stack\_operator;

void input(char t[]);  //输入表达式 每个操作数需小于10 操作数与操作数、运算符与运算符之间不相邻

void push\_num(Stack\_num \*num, char n);   //数字压栈

int pop\_num(Stack\_num \*num);   //数字出栈

void push\_operator(Stack\_operator \*oper, char n);  //运算符压栈

char pop\_operator(Stack\_operator \*oper);  //运算符出栈

int judgement(char s);  //判断运算符级别

int op(Stack\_operator \*oper, Stack\_num \*num, char sta[]);  //运算

int main(){

    char sta[STACK\_LEN];

    int result;

    Stack\_num \*num;

    Stack\_operator \*oper;

    num = (Stack\_num\*)malloc(STACK\_LEN);

    oper = (Stack\_operator\*)malloc(STACK\_LEN);

    num->top = 0;

    oper->top = 0;

    input(sta);

    result = op(oper, num, sta);

    printf("\nthe result is %d", result);

    return 0;

}

int op(Stack\_operator \*oper, Stack\_num \*num, char sta[]){

    int i = 0;

    while(oper->top == 0 || num->top == 0){   //在运算前，保证两个栈都不为空

        if(sta[i] > 47)

            push\_num(num, sta[i]);

        else

            push\_operator(oper, sta[i]);

        i++;

    }

    while(sta[i] != '\0'){

        if(sta[i] > 47)

            push\_num(num, sta[i]);

        else{

            if(oper->sta[oper->top - 1] == '(' || judgement(sta[i]) >= judgement(oper->sta[oper->top - 1]))

                push\_operator(oper, sta[i]);

            else{

                int a, b, c, d;

                a = pop\_num(num);

                b = pop\_num(num);

                d = pop\_operator(oper);

                if(d == '+')

                    c = b + a + 48;

                else if(d == '-')

                    c = b - a + 48;

                else if(d == '\*')

                    c = a \* b + 48;

                else if(d == '/')

                    c = b / a + 48;

                push\_num(num, c);

                if(oper->sta[oper->top - 1] == '(')

                    pop\_operator(oper);

                if(sta[i] != ')')

                    push\_operator(oper, sta[i]);

            }

        }

        i++;

    }

    while(oper->top != 0){

        int a, b, c, d;

        a = pop\_num(num);

        b = pop\_num(num);

        d = pop\_operator(oper);

        if(d == '+')

            c = b + a + 48;

        else if(d == '-')

            c = b - a + 48;

        else if(d == '\*')

            c = a \* b + 48;

        else if(d == '/')

            c = b / a + 48;

        push\_num(num, c);

    }

    return num->sta[0];

}

int judgement(char s){   //小于进行运算

    switch (s){

    case(')'):

        return 0;

    case('+'):

        return 1;

    case('-'):

        return 1;

    case('/'):

        return 2;

    case('\*'):

        return 3;

    case('('): break;

    default:

        printf("error!\n"), exit(-2);

    }

}

void input(char sta[]){

    printf("input expression(the total length is less than %d)\n", STACK\_LEN);  //对括号整体进行运算，运算符放到括号后

    scanf("%s", sta);

}

void push\_num(Stack\_num \*num, char n){   //n is sta[i]

    if(num->top == STACK\_LEN){

        printf("the stack\_num is full.\n");

        exit(-1);

    }

    else{

        num->sta[num->top] = n - 48;

        num->top++;

    }

}

int pop\_num(Stack\_num \*num){

    int n;

    if(num->top == 0){

        printf("the stack\_num is empty.\n");

        exit(-1);

    }

    else{

        n = num->sta[num->top - 1];

        num->top--;

    }

    return n;

}

void push\_operator(Stack\_operator \*oper, char n){

    if(oper->top == STACK\_LEN){

        printf("the stack\_operator is full.\n");

        exit(-1);

    }

    else{

        oper->sta[oper->top] = n;

        oper->top++;

    }

}

char pop\_operator(Stack\_operator \*oper){

    char n;

    if(oper->top == 0){

        printf("the stack\_operator is empty.\n");

        exit(-1);

    }

    else{

        n = oper->sta[oper->top - 1];

        oper->top--;

    }

    return n;

}