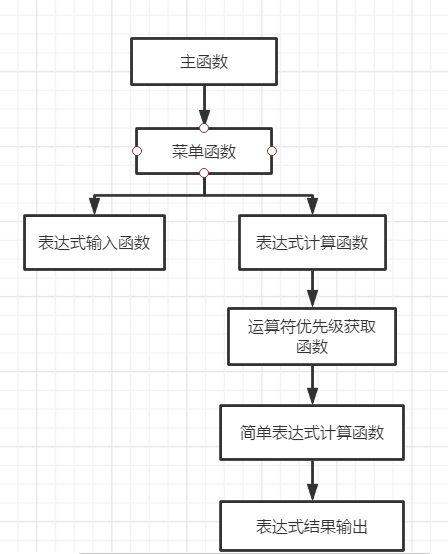
**数据结构实习报告**

**1.需求规格说明**

对一个合法的中缀表达式求值，表达式只包含+、-、×、÷四个双目运算符，且运算符本身不具有二义性。即用户输入带括号的表达式通过运算输出最后的计算结果。  
 **2.总体分析与设计**  
  
**(1)设计思想:**

存储结构：使用栈来保存计算过程中的运算符和操作数，使用vector容器来保存处理后的中缀表达式。

主要的算法思想：将用户输入的表达式进行拆分处理并保存到vector容器中以待计算时使用；对表达式进行计算时将表达式中的操作数和运算符依次压入操作数和运算符栈中，其中需要对表达式中的数字重新进行字符串转数字操作，当遇到左括号时将其先压入运算符栈中待遇到右括号操作结束时再将其弹出，运算时通过比较运算符优先级来分辨运算顺序，待运算符栈中元素全部弹出后输出操作数栈栈顶元素，即为运算结果。  
  
**(2)设计表示:**

  
  
**(3)详细设计表示:**

中缀表达式输入函数：

先将用户输入的表达式用char数组来存储，然后转成string类型变量。将string类型变量进行拼接操作然后存入一个vector容器中。

判断运算符优先级函数：

事先设定好每个运算符优先级对应的数字，使用if else语句来确定当前的运算符优先级对应的数字值。

计算简单表达式函数：

通过从运算符栈和操作数栈传递的参数，使用if else语句来确定进行什么样的计算并返回计算结果。

表达式求值函数：

定义一个运算符栈和一个操作数栈用来保存运算符和操作数，然后对保存表达式的vector容器进行遍历，将带数字的字符串转成数字，对左括号进行判别，如果后面跟的为“-”，将对相隔一个字符串的字符串进行判断：如果是“.”将其保存为一个负小数，否则保存为负数，并将其压栈，进行后续字符串判别。然后经过判断运算符优先级对表达式进行计算。  
  
**3.编码**

问题：

使用vector迭代器对vector容器进行遍历时经常出现溢出的问题、对负数进行识别尤其是负小数的识别时遇到了思路障碍

解决方法：

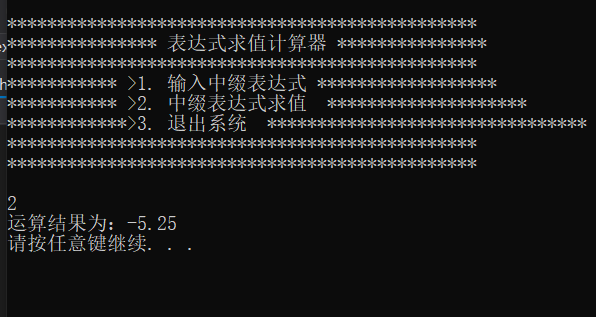
使用单步调试（debug）的方法对程序单步执行，根据每一步的变量值改变情况来继续编码；对于思路障碍，在纸上写出可能可行的方法然后逐一编码实现并单步调试进行修改。

**4.程序及算法分析**

使用说明：

运行程序后，根据提示输入想使用的功能（注意：需要先输入中缀表达式才能进行计算），输入中缀表达式后回到主菜单选择计算结果，进行计算后结果输出在控制台上。

程序运行结果：



（图为对3.5\*(7+2)/(-6)计算后的结果）

讨论与分析：

本次实验考察了对栈的操作和理解以及对字符串处理的方法选择，需要对栈有较为深刻的理解以及对栈的原理进行掌握才能编写出算法并编码实现，这次实验使我受益良多。

时空复杂度：

时间复杂度：O(n)；空间复杂度：O(n)  
  
**5.小结**

经验与体会：

通过单步调试，对栈的工作原理有了深刻的理解；通过vector迭代器的使用，对vector容器的使用更加得心应手；栈是先进后出的特性可以被充分利用于表达式的计算中。通过本次实习，我对栈操作更加熟悉了，对栈的理解更加深刻。

需要改进的地方：

计算表达式的代码还有些冗长，可以更加简洁。  
 **6.附录**

int Caculate::get\_Value(string operation) { //获取元素优先级

int value = 0;

if (operation == "+") {

value = ADD;

}

else if (operation == "-") {

value = SUB;

}

else if (operation == "\*") {

value = MUL;

}

else if (operation == "/") {

value = DIV;

}

/\*else if (operation == "%") {

value = Demo;

}\*/

else {

value = 0;

}

return value;

}

float Caculate::result(float a, string str, float b) { //计算简单表达式

float result = 0;

if (str == "+") {

result = a + b;

}

else if (str == "-") {

result = a - b;

}

else if (str == "\*") {

result = a \* b;

}

else if (str == "/") {

result = a / b;

}

/\*else if (str == "%") {

result = a % b;

}\*/

else {

result = 0;

}

return result;

}

void Caculate::input\_Infix() { //输入中缀表达式

this->infix\_Expression.clear(); //清空容器，方便接收用户接下来的输入

system("cls");

cout << "请输入你想要计算的中缀表达式：" << endl;

char cha[50]; //用来接收将要输入的表达式, 最多接收50个字符

cin.ignore(); //cin.getline默认以换行符为结束标志,要消耗掉前面的换行符需要多调用一次即使用两次cin.getline()

cin.getline(cha, 50); //或者这之前使用cin.ignore();

string str = string(cha); //将字符数组强转成字符串

int len = (int)str.length(); //表达式长度

int i = 0;

do {

if (str[i] == '=' || str[i] == ' ') { //当为空格或者‘=’ 的时候，就直接跳过

i++;

}

else if (str[i] < 48 || str[i] > 57) { //当ascii码不在0-9之间，就直接保存

string temp = "";

temp += str[i];

this->infix\_Expression.push\_back(temp);

i++;

}

else {

string str\_temp = "";

while (i < len && str[i] >= 48 && str[i] <= 57) { //当在0-9之间就拼接成原来的数字，然后再保存

str\_temp += str[i];

i++;

}

this->infix\_Expression.push\_back(str\_temp);

}

}

while (i < len);

this->is\_Input = true;

cout << "===========================" << endl;

cout << "表达式输入成功！:)" << endl;

cout << endl;

system("pause");

system("cls");

}

void Caculate::caculate() { //中缀表达式求值

//定义两个栈，一个，一个

stack<string> POTR; //运算符栈

stack<double> OPND; //操作数栈

vector<string>::iterator temp;//

for (vector<string>::iterator it = this->infix\_Expression.begin(); it != this->infix\_Expression.end(); it++) {

temp = it;//

string str = \*it;

string tempstr = \*temp;//

stringstream ss;

float num = 0;

if (str[0] >= '0' && str[0] <= '9') { //如果第一个字符为数字字符，则这个字符串是数字，用istringsteam 转成数字

bool f = (temp != this->infix\_Expression.end()-1);

if (f==1) {

if (\*++temp == ".") {//

str += \*temp;

temp++;

str += \*temp;

istringstream istr(str); //这样就可以实现多位整数的运算

istr >> num;

OPND.push(num);

it++;

it++;

continue;

}

}

istringstream istr(str); //这样就可以实现多位整数的运算

istr >> num;

OPND.push(num);

}

else if (str == "(") { //如果遇到左括号，就压进运算符栈

//当前it在(处

if (\*++temp == "-") {

string t1 = \*temp;

temp++;//temp在小数点前的数

t1 += \*temp;

temp++;//temp为小数点

it++;//it在-处

if (\*temp == ".") {

t1 += \*temp;

temp++;

t1 += \*temp;

it++;//有小数点时在小数点前数字

it++;//指向小数点

}

it++;//如果括号内非小数则it在数字处

//如果括号内有小数则在小数点右侧数字

it++;//跳过右括号

istringstream istr(t1); //这样就可以实现多位整数的运算

istr >> num;

OPND.push(num);

continue;

}

POTR.push(str);

}

else if (str == ")") { //如果遇到右括号，就去匹配运算符栈中的左括号，当不匹配的时候，操作数栈弹出两个操作数，

while (POTR.top() != "(") { //和当前运算符栈的栈顶运算符进行运算，最后将结果保存到操作数栈

float a = OPND.top();

OPND.pop();

float b = OPND.top();

OPND.pop();

float result = this->result(b, POTR.top(), a);

OPND.push(result);

POTR.pop();

}

POTR.pop(); //弹出"("

}

else {

if (POTR.size() == 0 || POTR.top() == "(") {

POTR.push(str);

}

else {

while (POTR.size() != 0 && get\_Value(POTR.top()) >= get\_Value(str)) {

float a = OPND.top(); //当当前运算符优先级小于等于栈顶元素优先级时，就从操作数栈弹出两个操作数

OPND.pop(); //和栈顶运算符参与运算，将结果保存到操作数栈中

float b = OPND.top();

OPND.pop();

float result = this->result(b, POTR.top(), a);

OPND.push(result);

POTR.pop();

}

POTR.push(str); //之后将当前运算符压如运算符栈

}

}

}

while (POTR.size() != 0) { //最后当运算符栈不为空的时候，再依次弹出运算

float a = OPND.top();

OPND.pop();

float b = OPND.top();

OPND.pop();

float result = this->result(b, POTR.top(), a);

OPND.push(result);

POTR.pop();

}

cout << "运算结果为：" << OPND.top() << endl;

OPND.pop();

system("pause");

system("cls");

}