项目说明文档

数据结构课程设计

——算数表达式求解

作 者 姓 名： 肖杨

学 号： 1950430

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

1 分析 3

2 设计 3

3 实现 8

4 测试 17

1 分析

1.1 背景分析

日常生活中，算数表达式的计算是程序中最经常要处理的事件之一，所以编写算术表达式的计算程序是具有很重大的意义。一个算数表达式有前缀表示法，中缀表示法和后缀表示法等形式。日常使用的算术表达式是采用中缀表示法，即二元运算符位于两个运算数中间。但是由于中缀表达式中有操作符的优先级问题，还有可加括号改变运算顺序的问题，所以对于编译程序来说，一般不使用中缀表示处理表达式。解决办法是用后缀表达式表示。本项目需要设计程序将中缀表达式转换成为后缀表达式，并计算其最终结果。

1.2 功能分析

在项目的背景下，需要实现的功能是给定算数表达式判断其合法性，将中缀转化为后缀，以及后缀表达式的计算。

本程序支持long long 范围内的加减乘除、取余、乘方运算，并允许添加括号与单目运算符+、-。其中除法不允许除0，若不能整除则会向下取整，乘方不允许负数幂。计算过程中应保证表达式中不省略乘号且括号匹配，单目运算符只能在首端或左括号后出现，读取到第一个等号后忽略后继输入，且要求必须有等号。

任何不符合要求的表达式或计算结果溢出的表达式均会被指出错误并要求重新输入，从而保证输出的结果一定为正确结果。

2 设计

2.1 数据结构设计

如上功能分析所述，该系统要求将中缀表达式转化为后缀表达式，并计算后缀表达式。一般的实现是使用栈数据结构，不需要其他数据结构。

为了保证运算不出错，栈具有自动扩充功能，但实际上由于出入栈同时进行，对于大部分表达式，栈的大小一般在100以内。

2.2 类结构设计

为了保证设计的数据结构的泛用性，本项目选择将stack类设计为模板类。在运算过程中，首先需要保存表达式再进行运算，因此设计了answer类，。

2.3 成员与操作设计

栈类

template <class T>

class stack

{

public:

stack(int size);

~stack();

bool isEmpty();

bool isFull();

int stacklength();

bool push(T elem);

void expand();

void clear();

T pop();

T peek();

private:

T\* m\_buffer;

int m\_size;

int m\_element;

};

私有成员：

T\* m\_buffer;

//储存拥有连续地址的对象

int m\_size;

//储存栈大小

int m\_element;

//栈中已有对象数目

公有操作：

stack(int size);

//用栈大小进行初始化

bool isEmpty();

//判空

bool isFull();

//判满

int stacklength();

//返回已有元素数目

bool push(T elem);

//入栈

void expand();

//自动/手动扩充栈大小

void clear();

//清空栈

T pop();

//出栈并返回出栈元素

T peek();

//返回栈顶元素

Answer类

class answer

{

public:

answer();

~answer();

bool calculation();

void clear();

private:

string calc;

bool isError = false;

stack<long long>\* numbers;

stack<char>\* operators;

bool checkNumber(char elem);

void checkOperator(char elem);

void useOperator(int op);

bool plusCheck(const long long &ans, const long long &left, const long long &right);

bool minusCheck(const long long &ans, const long long &left, const long long &right);

bool mulityCheck(const long long &ans, const long long &left, const long long &right);

bool powerCheck(const long long &ans, const long long &left, const long long &right);

int getValue(char op);

};

私有成员：

string calc;

//储存表达式

bool isError = false;

//判断表达式与运算是否出错

stack<long long>\* numbers;

//用于储存数字的栈

stack<char>\* operators;

//用于储存运算符的栈

私有操作：

bool checkNumber(char elem);

//检查是否为数字，从而进行相应操作

void checkOperator(char elem);

//对运算符进行检查，从而实现入栈出栈与运算

void useOperator(int op);

//对于需要计算的运算符，进行运算

bool plusCheck(const long long &ans, const long long &left, const long long &right);

//检查加法溢出

bool minusCheck(const long long &ans, const long long &left, const long long &right);

//检查减法溢出

bool mulityCheck(const long long &ans, const long long &left, const long long &right);

//检查乘法溢出

bool powerCheck(const long long &ans, const long long &left, const long long &right);

//检查乘方溢出

int getValue(char op);

//返回运算符对应的值

公有操作：

answer();

//构造函数，进行初始化

bool calculation();

//进行计算，返回是否进行下一次运算

void clear();

//若进行下一次运算，再次初始化成员

2.4 系统设计

系统首先提示用户输入运算表达式，然后进行表达式的中缀转化后缀和计算，若表达式中有非法输入或计算结果出现溢出，提示用户重新输入。否则输出运算结果。

然后询问用户是否计算下一个表达式，并进行下一次运算或退出程序。

3 实现

3.1 输入表达式的实现

3.1.1 输入表达式并进行遍历的流程

输入表达式时，整个表达式的内容由字符串calc存储，随后对calc进行一遍遍历并转为后缀表达式，直到找到第一个等号为止，输出运算结果。

过程中利用isNumber来判断多位数的输入和数字入栈，并判断输入符号是否合法(如连续输入多个符号无数字)。

在输入过程中不断检查符号并进行计算。

3.1.2 核心代码

cout << "请输入一行算式" << endl;

getline(std::cin, calc);

long long temp = 0;

bool single = false;

bool isNumber = false;

char last = '=';

for (auto elem : calc)

{

if (checkNumber(elem))

{

if (isNumber)

{

if (temp == 0)

{

isError = true;

break;

}

if (temp \* 10 + elem > temp)

{

temp = temp \* 10 + elem - '0';

}

else

{

cout << "输入数溢出" << endl;

isError = true;

}

}

else//第一次得到数字直接进入else

{

temp = elem - '0';

isNumber = true;

}//得到数字

}

else

{

if (elem != '(' && (last == '=' || last == '('))//说明为单目运算符

{

if (single)

{

isError = true;

}

else

{

if (elem == '-')

{

numbers->push(0);

operators->push('-');

single = true;

}

else if (elem == '+')

{

single = true;

}

else

{

isError = true;

}

}

}

else

{

if (checkNumber(last))//除前括号以外，其他运算符前应为数字

{

single = false;

numbers->push(temp);

temp = 0;

isNumber = false;

checkOperator(elem);

}

else if (elem == '(' || last == ')')//前括号前可以是符号，后括号可以视为数字但不进行数字插入

{

checkOperator(elem);

}

else

{

isError = true;

}

}

}

last = elem;

if (isError)

{

break;

}

if (operators->stacklength() == 0)

{

break;

}

}

3.1.3 输入表达式额外说明

输入表达式时，若出现数字溢出则会进行报错，同时任何不合法的输入都会进行报错。为了避免cin与getline使用的流出现冲突，所有输入都使用string流。

3.2 中缀转后缀功能的实现

3.2.1 中缀转后缀算法描述

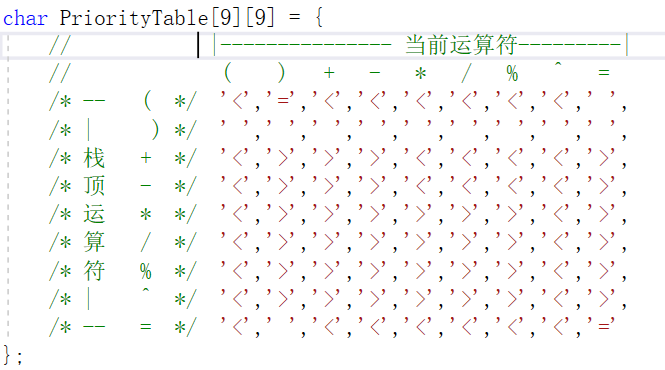
中缀转后缀的过程中，首先要对运算符优先级进行判断，同时要考虑到括号对优先级的改变。对于单目运算符，选择用一个局部变量single进行处理，保证只有一个单目运算符。正号将只被判断一次，负号将向numbers中加入0并向operators中加入-号，将取负变为减法，从而简化操作。

当前运算符优先级较大时，进行入栈操作。

两个运算符优先级相等时，只执行出栈操作。

当前运算符优先级较小时，栈顶元素出栈并执行运算，并将当前运算符与新栈顶进行比较。

操作符优先级如下，栈底存放=号用于匹配末尾=号。



3.2.2 核心代码

void answer::checkOperator(char elem)

{

int val1 = 0, val2 = 0;

val1 = getValue(elem);

if (val1 == -1)

{

isError = true;

return;

}

val2 = getValue(operators->peek());

while (!isError)

{

if (PriorityTable[val2][val1] == '<')

{

operators->push(elem);

return;

}

else if (PriorityTable[val2][val1] == '=')

{

operators->pop();

return;

}

else if(PriorityTable[val2][val1] == '>')

{

useOperator(val2);

operators->pop();

val2 = getValue(operators->peek());//所有正确的输入中，等号都在栈底，一定能执行peek

}

else//空格说明为错误的输入

{

isError = true;

return;

}

}

}

int answer::getValue(char op)

{

switch (op)

{

case '(':

return 0;

case ')':

return 1;

case '+':

return 2;

case '-':

return 3;

case '\*':

return 4;

case '/':

return 5;

case '%':

return 6;

case '^':

return 7;

case '=':

return 8;

default:

return -1;

}

}

3.3 计算后缀表达式功能的实现

3.3.1 计算后缀表达式过程

每次判断输入符号时，都可能执行运算。利用符号的不同值来执行相应操作，并且执行时要判断是否溢出。

由于回执行的运算符均为双目运算符，因此首先取得栈顶两个数字（输入操作已经保证一定有两个数字），取得预算结果后再入栈。

全部执行完毕后，若表达式未出错且此时符号栈为空，数字栈仅有一个数字，则进行输出结果。

3.3.2 核心代码

void answer::useOperator(int op)

{

auto right = numbers->pop(), left = numbers->pop();

long long ans = 0;

switch (op)

{

case 2:

ans = left + right;

if (plusCheck(ans, left, right))

{

numbers->push(ans);

}

else

{

cout << "加法溢出" << endl;

isError = true;

}

break;

case 3:

ans = left - right;

if (minusCheck(ans, left, right))

{

numbers->push(ans);

}

else

{

cout << "减法溢出" << endl;

isError = true;

}

break;

case 4:

ans = left \* right;

if (mulityCheck(ans, left, right))

{

numbers->push(ans);

}

else

{

cout << "乘法溢出" << endl;

isError = true;

}

break;

case 5:

if (right == 0)

{

cout << "禁止除零" << endl;

isError = true;

break;

}

ans = left / right;

if (ans == LLONG\_MIN && right == -1)

{

cout << "除法溢出" << endl;

isError = true;

break;

}

numbers->push(ans);

break;

case 6:

if (right == 0)

{

cout << "禁止除零" << endl;

isError = true;

break;

}

numbers->push(left%right);

break;

case 7:

if (right < 0)

{

cout << "不支持负数幂" << endl;

isError = true;

break;

}

ans = pow(left, right);

if (powerCheck(ans, left, right))

{

numbers->push(ans);

}

else

{

cout << "幂运算溢出" << endl;

isError = true;

}

break;

default:

break;

}

}

bool answer::calculation():

……

if (!isError && numbers->stacklength() == 1 && operators->isEmpty())

{

cout << "计算结果为：" << numbers->pop() << endl;

}

else

{

cout << "输入的表达式不合法，请重新输入" << endl;

}

……

3.4 总体系统实现

3.4.1 总体系统流程

首先利用构造函数初始化，随后在answer对象中进行输入计算，并判断是否继续运算。

3.4.2 核心代码

int main()

{

answer ans;

while (ans.calculation())

{

cout<< "进行下一次计算" << endl;

ans.clear();

}

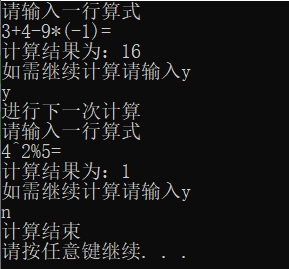
cout << "计算结束" << endl;

system("pause");

return 0;

}

3.4.3 系统实例



4 测试

4.1 功能测试

4.1.1 加法功能测试

**测试用例**：99+99=

**预期结果**：

198

**实验结果：**



4.1.2 减法功能测试

**测试用例：**1-99=

**预期结果：**

-98

**实验结果：**



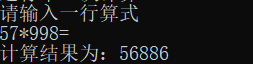
4.1.3 乘法功能测试

**测试用例：**57\*998=

**预期结果：**

56886

**实验结果：**



4.1.4 除法功能测试

**测试用例：**42/3=

**预期结果：**

14

**实验结果：**



4.1.5 取余功能测试

**测试用例：**97%5=

**预期结果：**

2

**实验结果：**



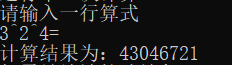
4.1.6 乘方功能测试

**测试用例：**3^2^4=

**预期结果：**

**实验结果：**

43046721



4.1.7 单目运算符测试

**测试用例：**-14+2=

**预期结果：**

-12

**实验结果：**



**测试用例：**+14\*(+22)=

**预期结果：**

308

**实验结果：**



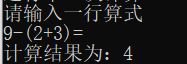
4.1.8 括号测试

**测试用例：**9-(2+3)=

**预期结果：**

4

**实验结果：**



4.1.9 综合测试

**测试用例：-**9-(2+3+(-99)\*13)+1598/(5+2^2^2)+99%21=

**预期结果：**

1364

**实验结果：**



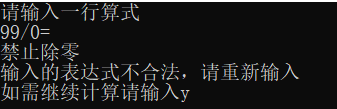
4.2 边界测试

4.2.1 除0

**测试用例：**99/0=

**预期结果：**给出错误提示，程序运行正常不崩溃。

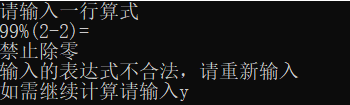
**实验结果：**



**测试用例：**99%(2-2)=

**预期结果：**给出错误提示，程序运行正常不崩溃。

**实验结果：**

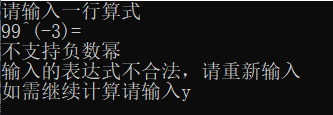


4.2.2 负数幂

**测试用例：**99^(-3)=

**预期结果：**给出错误提示，程序运行正常不崩溃。

**实验结果：**

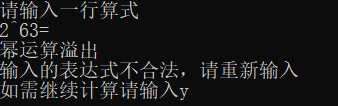


4.2.3 计算溢出

**测试用例：**2^63=

**预期结果：**给出错误提示，程序运行正常不崩溃。

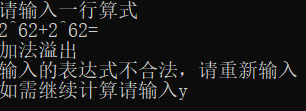
**实验结果：**



**测试用例：**2^62+2^62=

**预期结果：**给出错误提示，程序运行正常不崩溃。

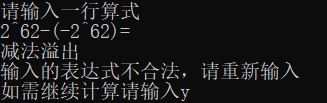
**实验结果：**



**测试用例：**2^62-(-2^62)=

**预期结果：**给出错误提示，程序运行正常不崩溃。

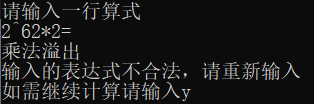
**实验结果：**



**测试用例：**2^62\*2=

**预期结果：**给出错误提示，程序运行正常不崩溃。

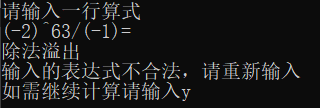
**实验结果：**



**测试用例：**(-2)^63/(-1)=

**预期结果：**给出错误提示，程序运行正常不崩溃。

**实验结果：**



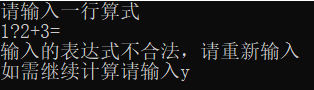
4.3 错误测试

4.3.1 输入错误的操作符

**测试用例：**1?2+3=

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

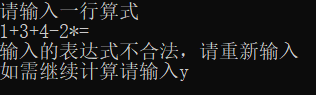


4.3.2 操作符与操作数不匹配

**测试用例：**1+3+4-2\*=

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

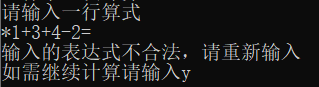
**实验结果：**



**测试用例：**\*1+3+4-2=

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

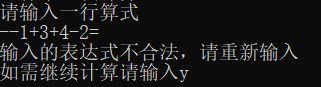
**实验结果：**



**测试用例：**--1+3+4-2=

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

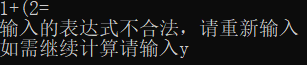


4.3.3 括号不匹配

**测试用例：**1+(2=

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

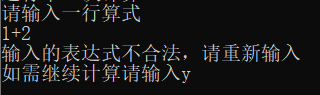


4.3.4 无等号/等号后有输入

**测试用例：**1+2

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

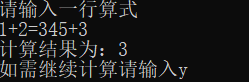
**实验结果：**



**测试用例：**1+2=345+3

**预期结果：**忽略等号后的内容，输出3

**实验结果：**

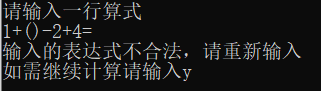


4.3.5 空括号

**测试用例：**1+()-2+4=

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**



4.3.6 表达式为空

**测试用例：**=

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

