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： 陈翔飞

学 号： 1851756

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 4](#_Toc27494243)

[1.1 背景分析 4](#_Toc27494244)

[1.2 功能分析 4](#_Toc27494245)

[2 设计 4](#_Toc27494246)

[2.1 数据结构设计 4](#_Toc27494247)

[2.2 类结构设计 4](#_Toc27494248)

[2.3 成员与操作设计 4](#_Toc27494249)

[2.4 系统设计 10](#_Toc27494250)

[3 实现 10](#_Toc27494251)

[3.1 中缀转后缀功能的实现 10](#_Toc27494252)

[3.1.1 中缀转后缀算法描述 10](#_Toc27494253)

[3.1.2 中缀转后缀功能核心代码 11](#_Toc27494254)

[3.2 计算后缀表达式功能的实现 12](#_Toc27494255)

[3.2.1 计算后缀表达式算法描述 12](#_Toc27494256)

[3.2.2 计算后缀表达式核心代码 12](#_Toc27494257)

[3.3 总体系统的实现 16](#_Toc27494258)

[3.3.1 总体系统流程图 16](#_Toc27494259)

[3.6.2 总体系统核心代码 16](#_Toc27494260)

[3.6.3 总体系统截屏示例 17](#_Toc27494261)

[4 测试(手动测试时请务必使用英文括号) 18](#_Toc27494262)

[4.1 功能测试 18](#_Toc27494263)

[4.1.1 加法功能测试 18](#_Toc27494264)

[4.1.2 减法功能测试 18](#_Toc27494265)

[4.1.3 乘法功能测试 18](#_Toc27494266)

[4.1.4 除法功能测试 19](#_Toc27494267)

[4.1.5 取余功能测试 19](#_Toc27494268)

[4.1.6 一元运算符测试 19](#_Toc27494269)

[4.1.7 乘方运算符测试 20](#_Toc27494270)

[4.1.8 括号运算符测试 20](#_Toc27494271)

[4.1.9 无括号的综合测试 21](#_Toc27494272)

[4.1.10 带括号的综合测试 21](#_Toc27494273)

[4.2 边界测试 21](#_Toc27494274)

[4.2.1 除0数据 21](#_Toc27494275)

[4.2.2 大整数加减法 22](#_Toc27494276)

[4.2.3 大整数乘除法 22](#_Toc27494277)

[4.2.4 大整数乘方 23](#_Toc27494278)

[4.2.5 大整数除0 24](#_Toc27494279)

[4.2.6 表达式缺末尾= 24](#_Toc27494280)

[4.3 出错测试 25](#_Toc27494281)

[4.3.1 输入范围外的操作符 25](#_Toc27494282)

[4.3.2 括号不匹配 25](#_Toc27494283)

[4.3.3 操作数和操作符不匹配 25](#_Toc27494284)

[4.3.4 空的括号表达式 26](#_Toc27494285)

[4.3.5 空的表达式 26](#_Toc27494286)

[4.3.6 多余的= 27](#_Toc27494287)

# 1 分析

## 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。在本项目中，表达式支持加减乘除，取余，乘方运算，还支持表达式中添加括号，以及一元运算符+，-。

# 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，这里栈采取数组实现，故还需要数据结构Vector。

## 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Vector类和Stack类都设计为模板类。为了判断表达式合法性，故需要先将表达式存储下来，所以还需要设计Expression类，为了将表达式中的运算符和数字统一放入Expression类，还设计了ExpressionElement类。同时，为了支持大整数运算，还设计了BigInteger类。

## 2.3 成员与操作设计

**向量类（Vector）**

**类定义：**

1. **template**<**typename** ElementType> **class** Vector
2. {
3. **public**:
4. ~Vector<ElementType>();
5. Vector<ElementType>() = **default**;
6. Vector<ElementType>(**const** Vector<ElementType> & v);
7. Vector<ElementType>& operator = (**const** Vector<ElementType>&v);
8. **void** PushBack(**const** ElementType& t);
9. **void** PopBack();
10. **void** Clear();
11. **int** GetSize() **const**;
12. **void** ReSize(**int** NewSize);
13. ElementType& operator[](**int** Index) **const**;
14. **bool** Empty()**const**;
15. **const** ElementType& Back() **const**;
16. **private**:
17. **void** Extend();
18. **int** Size = 0;
19. **int** Capacity = 0;
20. ElementType\* Array = nullptr;
21. };

**私有成员：**

int Size;//Vector中实际储存的元素数量

int Capacity;//Vector已经申请的空间

ElementType\* Array;//储存的数据的起始地址

**私有操作：**

void Extend();

//扩容函数，当容量不足时调用

**公有操作：**

~Vector<ElementType>();

//析构函数，通过调用Clear()来释放内存

Vector<ElementType>() = default;

//默认构造函数

Vector<ElementType>(const Vector<ElementType> & v);

//拷贝构造函数

Vector<ElementType>& operator = (const Vector<ElementType>&v);

//重载=运算符，使该类支持赋值运算

void PushBack(const ElementType& t);

//向Vector末尾添加一个元素

void PopBack();

//删除末尾的元素

void Clear();

//清空Vector，释放内存

int GetSize() const;

//返回储存元素的数量

void ReSize(int NewSize);

//重设Vector的大小

ElementType& operator[](int Index) const;

//重载[]运算符，使Vector可以像数组一样使用

bool Empty()const;

//判断Vector是否为空

const ElementType& Back() const;

//返回末尾的元素

**栈类（Stack）**

**类定义：**

1. **template** <**typename** ElementType> **class** Stack
2. {
3. **public**:
4. **void** Push(ElementType E);
5. **void** Pop();
6. ElementType Top();
7. **bool** Empty();
8. **int** Size();
9. **private**:
10. Vector<ElementType> L;
11. };

**私有成员：**

Vector<ElementType> L;//储存栈内的元素

**公有操作：**

void Push(ElementType E);

//向栈顶添加元素

void Pop();

//弹出栈顶元素

ElementType Top();

//返回栈顶元素

bool Empty();

//判断栈是否为空

int Size();

//返回栈的大小

**大整数类（BigInteger）**

**类定义：**

1. **class** BigInteger
2. {
3. **public**:
4. **explicit** BigInteger();
5. BigInteger(**const** **int** i);
6. BigInteger(**const** string& strValues);
7. BigInteger(**const** BigInteger& bigInt);
8. ~BigInteger() {};
9. **void** setValue(**const** string& strValues);
10. **bool** **inline** isPositive() { **return** flag; }
11. **bool** **inline** isZero() { **return** values == "0"; }
12. **int** compareBigint(**const** BigInteger& rhs)**const**;
13. **const** BigInteger absolute()**const**;
14. BigInteger& operator = (**const** BigInteger& rhs);
16. **friend** ostream& operator <<(ostream& ou, **const** BigInteger& bigInt);
17. **friend** istream& operator >>(istream& in, BigInteger& bigInt);
18. **friend** **const** **bool** operator >(**const** BigInteger& lhs, **const** BigInteger& rhs);
19. **friend** **const** **bool** operator ==(**const** BigInteger& lhs, **const** BigInteger& rhs);
20. **friend** **const** BigInteger operator + (**const** BigInteger& lhs, **const** BigInteger& rhs);
21. **friend** **const** BigInteger operator - (**const** BigInteger& lhs, **const** BigInteger& rhs);
22. **friend** **const** BigInteger operator \* (**const** BigInteger& lhs, **const** BigInteger& rhs);
23. **friend** **const** BigInteger operator / (**const** BigInteger& lhs, **const** BigInteger& rhs);
24. **friend** **const** BigInteger operator % (**const** BigInteger& lhs, **const** BigInteger& rhs);
25. **friend** **const** BigInteger operator ^(**const** BigInteger& lhs, **const** BigInteger& rhs);

28. **private**:
29. string values;
30. **bool** flag;
31. };

**私有成员：**

string values;//用字符串储存的整数

bool flag;//整数的符号位

**公有操作：**

explicit BigInteger();

//默认构造函数

BigInteger(const int i);

//参数为int的构造函数

BigInteger(const string& strValues);

//参数为string的构造函数

BigInteger(const BigInteger& bigInt);

//拷贝构造函数

~BigInteger() {};

//析构函数

void setValue(const string& strValues);

//更改该大整数的值

bool inline isPositive() { return flag; }

//判断该大整数是否为非负数

bool inline isZero() { return values == "0"; }

//判断该大整数是否为0；

int compareBigint(const BigInteger& rhs)const;

//比较两个大整数

const BigInteger absolute()const;

//返回该大整数的绝对值

BigInteger& operator = (const BigInteger& rhs);

//重载=运算符，使该类支持赋值运算

friend ostream& operator <<(ostream& ou, const BigInteger& bigInt);

//重载<<运算符，使该类支持<<运算

friend istream& operator >>(istream& in, BigInteger& bigInt);

//重载>>运算符，使该类支持>>运算

friend const bool operator >(const BigInteger& lhs, const BigInteger& rhs);

//重载>运算符，使该类支持>运算

friend const bool operator ==(const BigInteger& lhs, const BigInteger& rhs);

//重载==运算符，使该类支持==运算

friend const BigInteger operator + (const BigInteger& lhs, const BigInteger& rhs);

//重载+运算符，使该类支持+运算

friend const BigInteger operator - (const BigInteger& lhs, const BigInteger& rhs);

//重载-运算符，使该类支持-运算

friend const BigInteger operator \* (const BigInteger& lhs, const BigInteger& rhs);

//重载\*运算符，使该类支持\*运算

friend const BigInteger operator / (const BigInteger& lhs, const BigInteger& rhs);

//重载/运算符，使该类支持/运算

friend const BigInteger operator % (const BigInteger& lhs, const BigInteger& rhs);

//重载%运算符，使该类支持%运算

friend const BigInteger operator ^(const BigInteger& lhs, const BigInteger& rhs);

//重载^运算符，使该类支持^运算

**表达式元素类（ExpressionElement）**

**类定义：**

1. **struct** ExpressionElement
2. {
3. **bool** IsNum=**false**;
4. BigInteger Num;
5. **int** Symbol = 0;
6. };

**公有成员：**

bool IsNum;//该元素是否为数字的标志符

BigInteger Num;//储存数字

int Symbol = 0;//储存符号

**表达式类（Expression）**

**类定义：**

1. **class** Expression
2. {
3. **public**:
4. **void** MakeStringToExpression(**const** string& s);
5. **void** CalculatePost();
6. **void** CalculateExpression();
7. BigInteger GetResult();
8. **bool** IsCorrect = **true**;
9. **private**:
11. Vector<ExpressionElement>  Infix, Postfix;
12. BigInteger Result;
13. };

**私有成员：**

Vector<ExpressionElement> Infix;//储存前缀表达式

Vector<ExpressionElement> Postfix;//储存后缀表达式

BigInteger Result;//储存表达式计算结果

**公有成员：**

bool IsCorrect = true; //标志表达式是否正确的标志符

**公有操作：**

void MakeStringToExpression(const string& s);

//将字符串转化中缀表达式

void CalculatePost();

//将中缀表达式转化为后缀表达式

void CalculateExpression();

//计算后缀表达式的结果

BigInteger GetResult();

//返回结果

## 2.4 系统设计

系统首先调提示用户输入表达式，然后进行表达式的转化和计算。如果表达式出错，就提示用户重新输入；否则就输出表达式结果。然后询问用户是否计算下一个表达式。

# 3 实现

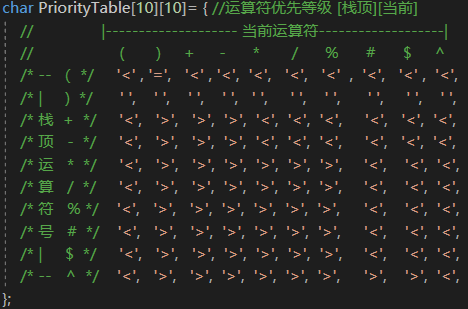
## 3.1 中缀转后缀功能的实现

### 3.1.1 中缀转后缀算法描述

由于中缀表达式中的操作符具有优先级，并且括号的出现会改变优先级，如此以来要将其转换成后缀表达式，我们需要考虑表达式中符号的优先级顺序以及有无括号的出现。

表达式的内容栈来存储，如果输入的是操作数，则可以直接放入Postfix中，如果是操作符，需要考虑优先级。

操作符的优先级分为进栈前和进栈后两种情况：（图上的‘#’和‘$’分别表示一元运算符+和-）



如果操作符ch的栈外优先级大于栈顶操作符的栈内优先级，则让该操作符ch进栈，并读入下一个表达式元素。如果操作符ch的栈外优先级小于栈顶操作符的栈内优先级，则让栈顶操作符退栈并且放入Postfix。如果操作符ch的栈外优先级等于栈顶操作符的栈内优先级，则让栈顶操作符退栈但不放入Postfix。

### 3.1.2 中缀转后缀功能核心代码

1. **inline** **void** Expression::CalculatePost()
2. {
3. **if** (!IsCorrect) **return**;
4. Postfix.Clear();
5. Stack<ExpressionElement> s;
6. **int** i;
7. **for** (i = 0; i < Infix.GetSize(); ++i)
8. {
9. **if** (Infix[i].IsNum) Postfix.PushBack(Infix[i]);
10. **else**
11. {
12. **if** (s.Empty()) s.Push(Infix[i]);
13. **else**
14. {
15. **switch** (PriorityTable[s.Top().Symbol][Infix[i].Symbol])
16. {
17. **case** '<':
18. s.Push(Infix[i]);
19. **break**;
20. **case** '>':
21. **while** (!s.Empty() && PriorityTable[s.Top().Symbol][Infix[i].Symbol] == '>')
22. {
23. Postfix.PushBack(s.Top());
24. s.Pop();
25. }
26. **if** (Infix[i].Symbol == 1)//当前为右括号
27. {
28. **if** (s.Empty() || s.Top().Symbol != 0)//未找到对应的左括号
29. {
30. cout << "括号不匹配！";
31. **goto** ERROR;
32. }
33. **else** s.Pop();//找到左括号，弹出
34. }
35. **else** s.Push(Infix[i]);
36. **break**;
37. **case** '=':
38. cout << "出现空的括号表达式！";
39. **goto** ERROR;
40. **break**;
41. **default**:
42. cout << "未知错误！";
43. **goto** ERROR;
44. }
45. }
46. }
47. }
48. **while** (!s.Empty())
49. {
50. **if** (s.Top().Symbol == 0)
51. {
52. cout << "括号不匹配！";
53. **goto** ERROR;
54. }
55. Postfix.PushBack(s.Top());
56. s.Pop();
57. }
58. **return**;
59. ERROR:
60. IsCorrect = **false**;
61. }

## 3.2 计算后缀表达式功能的实现

### 3.2.1 计算后缀表达式算法描述

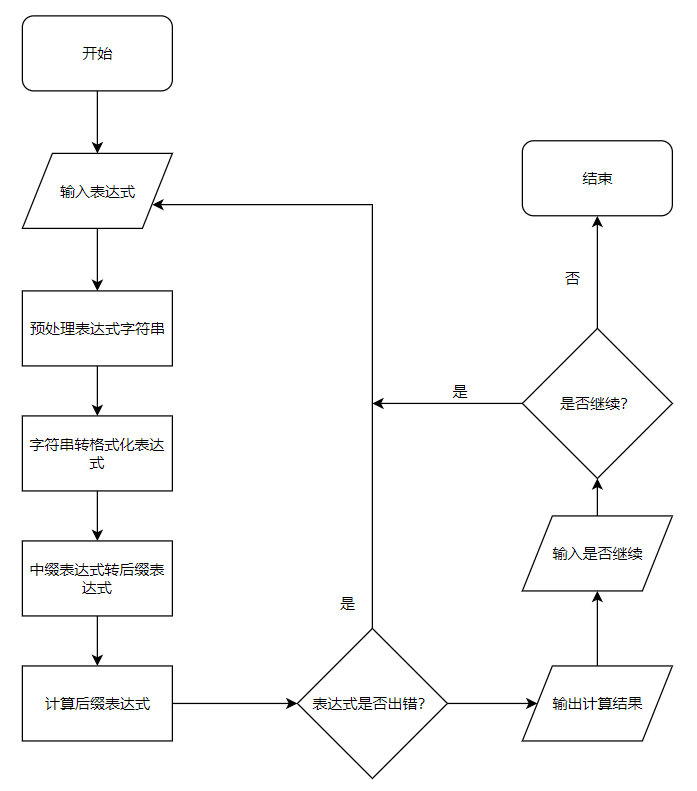
设置一个栈，开始时，栈为空，然后从左到右扫描后缀表达式，若遇操作数，则进栈；若遇运算符，则从栈中退出两个元素，先退出的放到运算符的右边，后退出的放到运算符左边，运算后的结果再进栈，直到后缀表达式扫描完毕。此时，栈中仅有一个元素，即为运算的结果。

### 3.2.2 计算后缀表达式核心代码

1. **inline** **void** Expression::CalculateExpression()
2. {
3. CalculatePost();
4. **if** (!IsCorrect) **return**;
5. Stack<BigInteger> ans;
6. **for** (**int** i = 0; i < Postfix.GetSize(); ++i)
7. {
8. **if** (Postfix[i].IsNum) ans.Push(Postfix[i].Num);
9. **else**
10. {
11. BigInteger t1, t2;
12. **switch** (Postfix[i].Symbol)
13. {
14. **case** 2:
15. **if** (ans.Empty())
16. {
17. cout << "操作符数与数字数量不匹配！";
18. **goto** ERROR;
19. }
20. t1 = ans.Top(); ans.Pop();
21. **if** (ans.Empty())
22. {
23. cout << "操作符数与数字数量不匹配！";
24. **goto** ERROR;
25. }
26. t2 = ans.Top(); ans.Pop();
27. ans.Push(t2 + t1);
28. **break**;
29. **case** 3:
30. **if** (ans.Empty())
31. {
32. cout << "操作符数与数字数量不匹配！";
33. **goto** ERROR;
34. }
35. t1 = ans.Top(); ans.Pop();
36. **if** (ans.Empty())
37. {
38. cout << "操作符数与数字数量不匹配！";
39. **goto** ERROR;
40. }
41. t2 = ans.Top(); ans.Pop();
42. ans.Push(t2 - t1);
43. **break**;
44. **case** 4:
45. **if** (ans.Empty())
46. {
47. cout << "操作符数与数字数量不匹配！";
48. **goto** ERROR;
49. }
50. t1 = ans.Top(); ans.Pop();
51. **if** (ans.Empty())
52. {
53. cout << "操作符数与数字数量不匹配！";
54. **goto** ERROR;
55. }
56. t2 = ans.Top(); ans.Pop();
57. ans.Push(t2 \* t1);
58. **break**;
59. **case** 5:
60. **if** (ans.Empty())
61. {
62. cout << "操作符数与数字数量不匹配！";
63. **goto** ERROR;
64. }
65. t1 = ans.Top(); ans.Pop();
66. **if** (ans.Empty())
67. {
68. cout << "操作符数与数字数量不匹配！";
69. **goto** ERROR;
70. }
71. t2 = ans.Top(); ans.Pop();
72. **if** (t1.isZero())
73. {
74. cout << "发生除零操作！";
75. **goto** ERROR;
76. }
77. ans.Push(t2 / t1);
78. **break**;
79. **case** 6:
80. **if** (ans.Empty())
81. {
82. cout << "操作符数与数字数量不匹配！";
83. **goto** ERROR;
84. }
85. t1 = ans.Top(); ans.Pop();
86. **if** (ans.Empty())
87. {
88. cout << "操作符数与数字数量不匹配！";
89. **goto** ERROR;
90. }
91. t2 = ans.Top(); ans.Pop();
92. **if** (t1.isZero())
93. {
94. cout << "发生除零操作！";
95. **goto** ERROR;
96. }
97. ans.Push(t2 %t1);
98. **break**;
99. **case** 7:
100. **if** (ans.Empty())
101. {
102. cout << "操作符数与数字数量不匹配！";
103. **goto** ERROR;
104. }
105. t1 = ans.Top(); ans.Pop();
106. t2.setValue("1");
107. ans.Push(t2\*t1);
108. **break**;
109. **case** 8:
110. **if** (ans.Empty())
111. {
112. cout << "操作符数与数字数量不匹配！";
113. **goto** ERROR;
114. }
115. t1 = ans.Top(); ans.Pop();
116. t2.setValue("-1");
117. ans.Push(t2\*t1);
118. **break**;
119. **case** 9:
120. **if** (ans.Empty())
121. {
122. cout << "操作符数与数字数量不匹配！";
123. **goto** ERROR;
124. }
125. t1 = ans.Top(); ans.Pop();
126. **if** (ans.Empty())
127. {
128. cout << "操作符数与数字数量不匹配！";
129. **goto** ERROR;
130. }
131. t2 = ans.Top(); ans.Pop();
132. ans.Push(t2^t1);
133. **break**;
134. }
135. }
136. }
137. Result = ans.Top();
138. **return**;
139. ERROR:
140. IsCorrect = **false**;
141. }

## 3.3 总体系统的实现

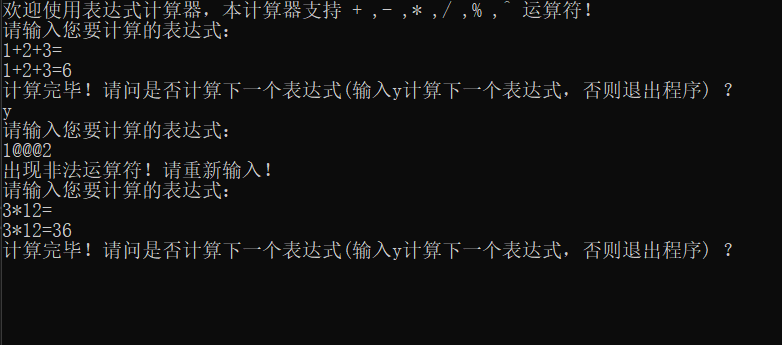
### 3.3.1 总体系统流程图



### 3.6.2 总体系统核心代码

1. **int** main(**void**)
2. {
3. **bool** IsLooping = **true**;
4. Expression E;
5. string ExpressionString;
6. cout << "欢迎使用表达式计算器，本计算器支持 + ,- ,\* ,/ ,% ,^ 运算符！" << endl;
7. **while** (IsLooping)
8. {
9. cout << "请输入您要计算的表达式：" << endl;
10. cin >> ExpressionString;
11. E.MakeStringToExpression(Pretreat(ExpressionString));
12. E.CalculateExpression();
13. **if** (E.IsCorrect)
14. {
16. cout << ExpressionString << "=" ;
17. cout << E.GetResult() << endl;
18. cout << "计算完毕！请问是否计算下一个表达式(输入y计算下一个表达式，否则退出程序) ？" << endl;
19. string Next;
20. cin >> Next;
21. **if** (Next != "y") IsLooping = **false**;
22. }
23. **else** cout << "请重新输入！" << endl;
24. }
25. **return** 0;
26. }

### 3.6.3 总体系统截屏示例



# 4 测试(手动测试时请务必使用英文括号)

## 4.1 功能测试

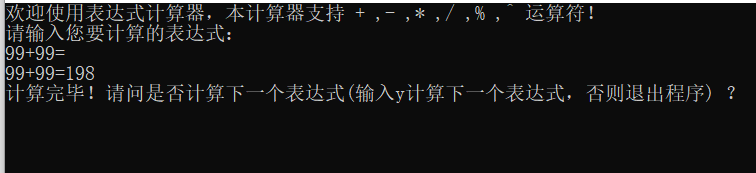
### 4.1.1 加法功能测试

**测试用例**：99+99=

**预期结果**：

99+99=198

**实验结果：**



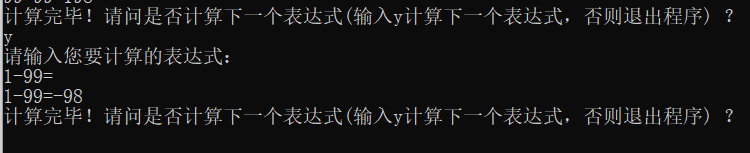
### 4.1.2 减法功能测试

**测试用例：**1-99=

**预期结果：**

1-99=-98

**实验结果：**



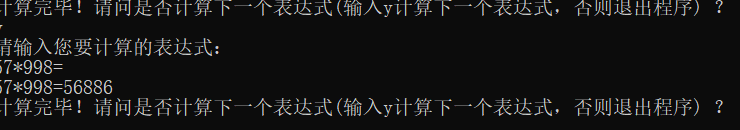
### 4.1.3 乘法功能测试

**测试用例：**57\*998=

**预期结果：**

57\*998=56886

**实验结果：**



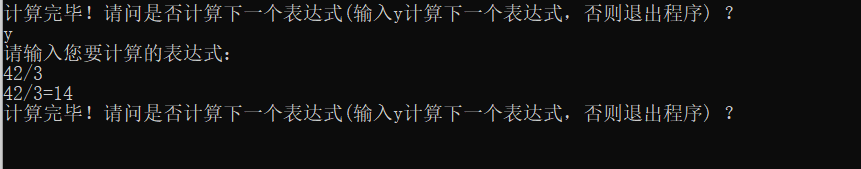
### 4.1.4 除法功能测试

**测试用例：**42/3=

**预期结果：**

42/3=14

**实验结果：**



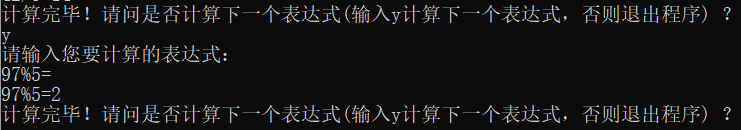
### 4.1.5 取余功能测试

**测试用例：**97%5=

**预期结果：**

97%5=

**实验结果：**



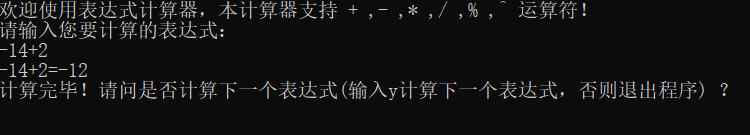
### 4.1.6 一元运算符测试

**测试用例：**-14+2=

**预期结果：**

-14+2=-12

**实验结果：**

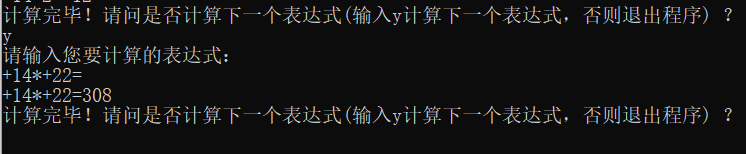


**测试用例：**+14\*+22=

**预期结果：**

+14\*+22=308

**实验结果：**



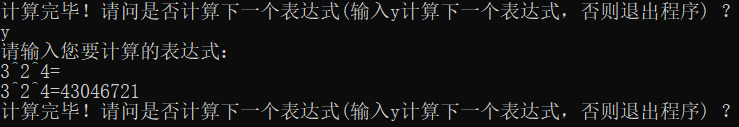
### 4.1.7 乘方运算符测试

**测试用例：**3^2^4=

**预期结果：**

3^2^4=43046721

**实验结果：**



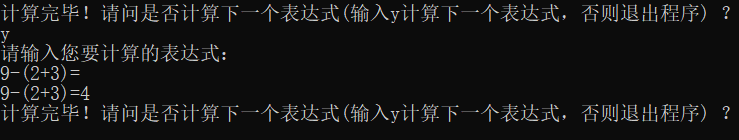
### 4.1.8 括号运算符测试

**测试用例：**9-(2+3)=

**预期结果：**

9-(2+3)=4

**实验结果：**



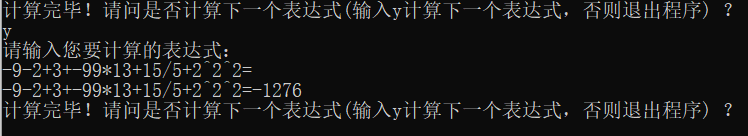
### 4.1.9 无括号的综合测试

**测试用例：-**9-2+3+-99\*13+15/5+2^2^2=

**预期结果：**

-9-2+3+-99\*13+15/5+2^2^2=-1276

**实验结果：**



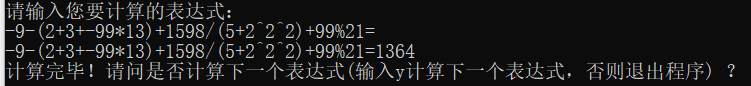
### 4.1.10 带括号的综合测试

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

**预期结果：**

-9-(2+3+-99\*13)+1598/(5+2^2^2)+99%21=1364

**实验结果：**



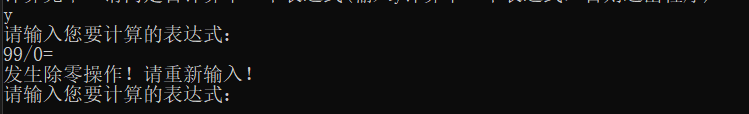
## 4.2 边界测试

### 4.2.1 除0数据

**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。

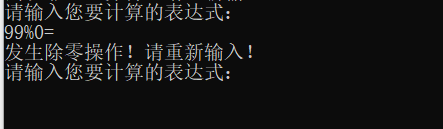
**实验结果：**



**测试用例：**99%0=

**预期结果：**给出错误提示，程序运行正常不崩溃。

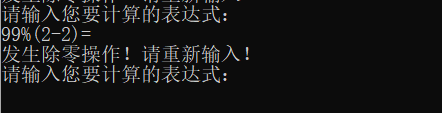
**实验结果：**



**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 大整数加减法

**测试用例：**

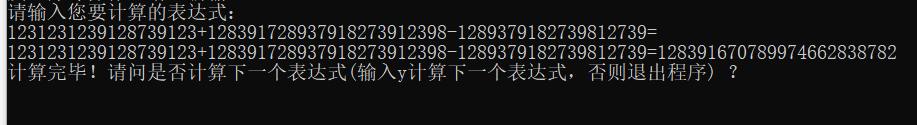
1231231239128739123+128391728937918273912398-1289379182739812739=

**预期结果：**

程序正常运行，输出：

1231231239128739123+128391728937918273912398-1289379182739812739=128391670789974662838782

**实验结果：**



### 4.2.3 大整数乘除法

**测试用例：**

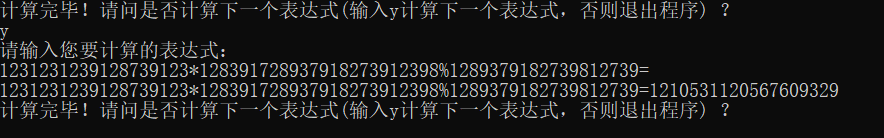
1231231239128739123\*128391728937918273912398%1289379182739812739=

**预期结果：**

程序正常运行，输出：

1231231239128739123\*128391728937918273912398%1289379182739812739=1210531120567609329

**实验结果：**



### 4.2.4 大整数乘方

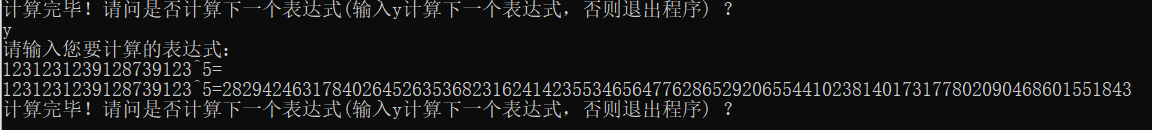
**测试用例：**1231231239128739123^5=

**预期结果：**

程序正常运行,输出：

1231231239128739123^5=2829424631784026452635368231624142355346564776286529206554410238140173177802090468601551843

**实验结果：**



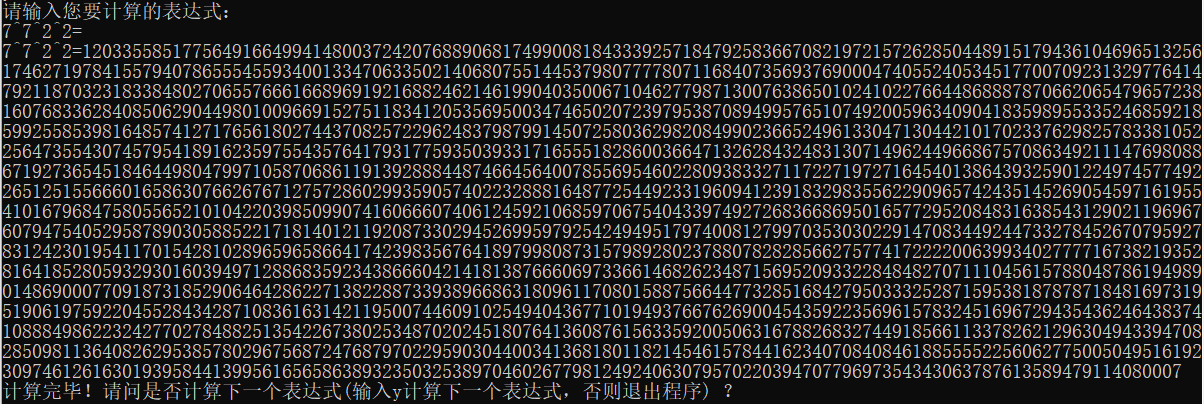
**测试用例：**7^7^2^2=

**预期结果：**

程序正常运行,输出：

7^7^2^2=12033558517756491664994148003724207688906817499008184333925718479258366708219721572628504489151794361046965132561746271978415579407865554559340013347063350214068075514453798077778071168407356937690004740552405345177007092313297764147921187032318338480270655766616689691921688246214619904035006710462779871300763865010241022766448688878706620654796572381607683362840850629044980100966915275118341205356950034746502072397953870894995765107492005963409041835989553352468592185992558539816485741271765618027443708257229624837987991450725803629820849902366524961330471304421017023376298257833810522564735543074579541891623597554357641793177593503933171655518286003664713262843248313071496244966867570863492111476980886719273654518464498047997105870686119139288844874664564007855695460228093833271172271972716454013864393259012249745774922651251556660165863076626767127572860299359057402232888164877254492331960941239183298355622909657424351452690545971619554101679684758055652101042203985099074160666074061245921068597067540433974927268366869501657729520848316385431290211969676079475405295878903058852217181401211920873302945269959792542494951797400812799703530302291470834492447332784526707959278312423019541170154281028965965866417423983567641897998087315798928023788078282856627577417222200639934027777167382193528164185280593293016039497128868359234386660421418138766606973366146826234871569520933228484827071110456157880487861949890148690007709187318529064642862271382288733938966863180961170801588756644773285168427950333252871595381878787184816973195190619759220455284342871083616314211950074460910254940436771019493766762690045435922356961578324516967294354362464383741088849862232427702784882513542267380253487020245180764136087615633592005063167882683274491856611337826212963049433947082850981136408262953857802967568724768797022959030440034136818011821454615784416234070840846188555522560627750050495161923097461261630193958441399561656586389323503253897046026779812492406307957022039470779697354343063787613589479114080007

**实验结果：**

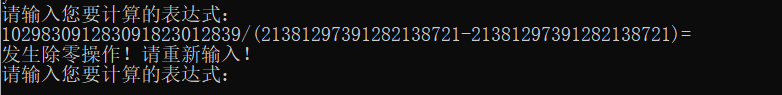


### 4.2.5 大整数除0

**测试用例：** 102983091283091823012839/(21381297391282138721-21381297391282138721)=

**预期结果：**程序正常运行，输出错误提示

**实验结果：**



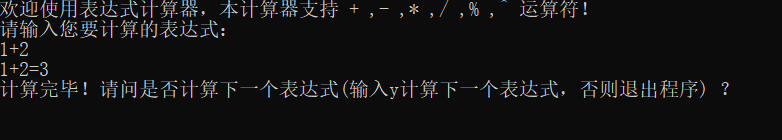
### 4.2.6 表达式缺末尾=

**测试用例：** 1+2

**预期结果：**程序正常运行，补齐=后输出：

1+2 =3

**实验结果：**



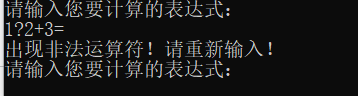
## 4.3 出错测试

### 4.3.1 输入范围外的操作符

**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：**

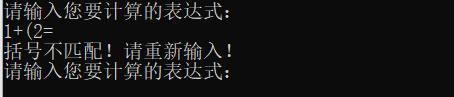


### 4.3.2 括号不匹配

**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：**

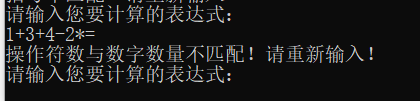


### 4.3.3 操作数和操作符不匹配

**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。

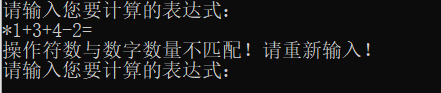
**实验结果：**



**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：**

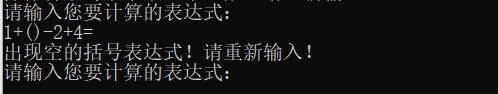


### 4.3.4 空的括号表达式

**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：**

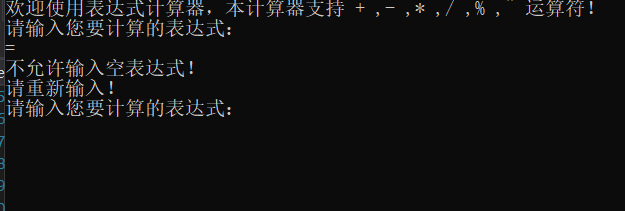


### 4.3.5 空的表达式

**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：**

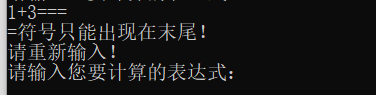


### 4.3.6 多余的=

**测试用例：**1+3===

**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=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：**

