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： 郑柯凡

学 号： 1950072

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](#_Toc13934)

[1.1背景分析 3](#_Toc23301)

[1.2功能分析 3](#_Toc25150)

[2 设计 3](#_Toc23308)

[2.1思路设计 3](#_Toc15586)

[2.2数据结构的选择 4](#_Toc2958)

[2.3类的设计 4](#_Toc30775)

[3 功能实现 7](#_Toc18596)

[3.1 后缀表达式转换功能的实现 7](#_Toc29395)

[3.1.1后缀表达式转换功能流程图 7](#_Toc17869)

[3.1.2后缀表达式转换功能核心代码 7](#_Toc11386)

[3.1.3后缀表达式转换功能截屏示例 8](#_Toc7444)

[3.2 计算后缀表达式功能的实现 8](#_Toc21267)

[3.2.1计算后缀表达式功能流程图 8](#_Toc30883)

[3.2.2计算后缀表达式功能核心代码 8](#_Toc21622)

[3.2.3计算后缀表达式功能截屏示例 10](#_Toc12774)

[3.3 计算器初始化及运行功能的实现 11](#_Toc911)

[3.3.1计算器初始化及运行功能流程图 11](#_Toc23192)

[3.3.2计算器初始化及运行功能核心代码 11](#_Toc5029)

[3.3.4计算器初始化及运行功能截屏示例 11](#_Toc1910)

[4测试 11](#_Toc710)

[4.1计算功能测试 12](#_Toc12665)

[4.2单目运算符计算功能测试 12](#_Toc11037)

[4.3表达式错误提醒功能测试 12](#_Toc21222)

**1 分析**

**1.1背景分析**

算术表达式是指数学课程中，由数字和运算符号组成的式子，可以简单清晰地记录或描述计算过程和内容。在计算机中，如何将表达式翻译成能够正确求值的指令序列，是处理程序要解决的基本问题。

**1.2功能分析**

从键盘上输入中缀算数表达式，包括括号，能够计算出表达式的值。若表达式有错，能给出适当的提示。支持包括加减，乘除取余，乘方和括号等操作符，其中优先级是等于<括号<加减<乘除取余<乘方。并且能处理单目运算符：+或-。

**2 设计**

**2.1思路设计**

中缀表达式是最常见的一种表达式书写形式，然而在计算机中，要求解中缀表达式需要用两个栈来实现，一个暂存操作数，一个暂存操作符。而用求解后缀表达式时只需要顺序扫描表达式即可，因此先考虑将输入的中缀表达式转换为后缀表达式。求解后缀表达式时，从左向右顺序扫描表达式，并使用一个栈暂存扫描到的操作数或计算结果。若扫描到的项为操作数，则将其压入栈中；若该项是操作符op，则从栈中退出两个操作数X和Y，形成运算指令XopY，并将计算结果重新压入栈中。当扫描完后缀表达式的所有项，若栈中没有元素或多于两个元素，则表达式有错；若栈中只有一个元素，则栈顶元素就是最后的计算结果。

**2.2数据结构的选择**

由于后缀表达式的计算中需要用到一个栈暂存操作数和运算结果，因此需要一个栈，包含压入，弹出，判空等操作。此外，每次都要对表达式的一项进行处理，因此采用有先进先出特性的队列来存放表达式。

**2.3类的设计**

一个结点类来存储表达式每一项的信息：其私有成员包括类型位type，数据位整型intnum和字符型charnum，若type为0，则intnum有效，若type为1，则charnum有效。以及栈内优先级isp和栈外优先级icp，用于将中缀表达式转换为后缀表达式。

结点类可作为栈和队列的数据成员。

1. //节点类，用于存储表达式中各个部分的值
2. **class** Node {
3. **friend** **class** LinkStack;
4. **friend** **class** Queue;
5. **friend** **class** Calculator;
6. **private**:
7. //存储的数据类型（0为整型，1为字符）
8. **bool** type;
9. **int** intnum;
10. **char** charnum;
11. //栈内优先级
12. **int** isp;
13. //栈外优先级
14. **int** icp;
15. //下一个结点地址
16. Node\* next;
17. **public**:
18. Node() { type = 0; intnum = 0; charnum = '#'; isp = 0; icp = 0; next = NULL; }
19. ~Node() {};
20. };

链式栈类，用链表构造的栈，便于插入删除等操作：其私有成员包括一个结点类，储存操作数和运算结果，以及size储存栈的大小。其公有成员包括压入，弹出和判空函数，其实现和链表的插入删除一致，之前的报告中也有提及，此处就不再贴代码赘述他们的实现了。

1. //栈
2. **class** LinkStack {
3. **friend** **class** Queue;
4. **friend** **class** Calculator;
5. **private**:
6. //栈顶元素地址
7. Node\* top;
8. //栈的大小
9. **int** size;
10. **public**:
11. //链式栈的构造函数(置空栈)
12. LinkStack() {
13. **this**->top = NULL;
14. **this**->size = 0;
15. };
16. //析构函数
17. ~LinkStack() {};
18. //将坐标放入栈顶
19. **void** Push(Node\* address);
20. //将栈顶的坐标弹出
21. **void** Pop();
22. //判断栈是否为空
23. **bool** IsEmpty() {
24. **if** (**this**->size > 0)
25. {
26. **return** 0;
27. }
28. **else**
29. {
30. **return** 1;
31. }
32. }
33. };

用链表实现的队列，用于存放表达式：其私有成员包括队列头和尾的地址，他们都是Node类型的指针，以及一个size储存队列的长度。其公有成员包括入队，出队，判空函数，以及转换后缀表达式的函数。其中，入队，出队函数的实现和链表的插入删除实现一致，此处就不再赘述了。

1. //队列，存放表达式
2. **class** Queue {
3. **friend** **class** Calculator;
4. **private**:
5. //队列头尾地址
6. Node\* front;
7. Node\* rear;
8. //队列的长度
9. **int** size;
10. **public**:
11. //队列的构造函数
12. Queue() {
13. front = rear = **new** Node; size = 1;
14. };
15. //析构函数
16. ~Queue() {};
17. //判断队列是否为空
18. **bool** IsEmpty() { **return** (size <= 1) ? 1 : 0; }
19. //添加节点
20. **void** EnQueue(Node\*& address);
21. //队头出列
22. **void** DeQueue();
23. //转换成后缀表达式
24. **void** Postfix(Node\* current);
25. };

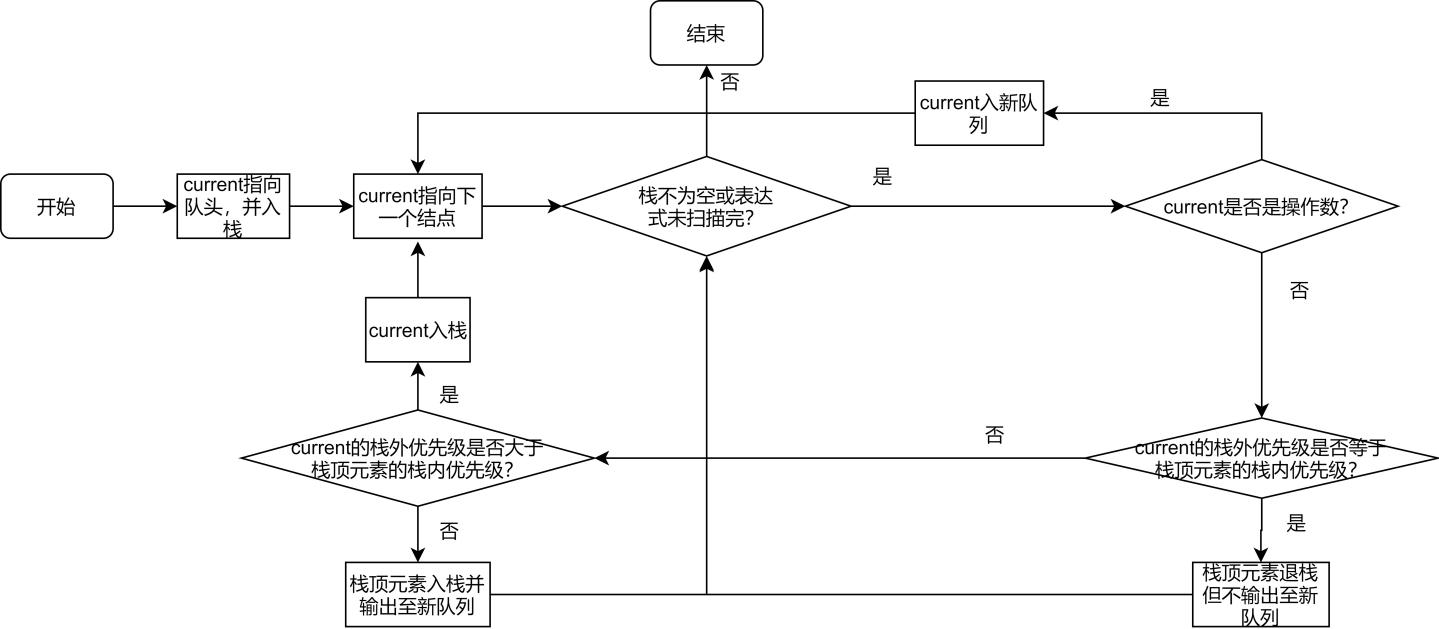
最后是计算器类，也是这个项目中的主体类：其私有成员包括一个队列，用来储存表达式；一个链式栈，用来储存计算过程中的操作数和最终结果；输入，运行，清空表达式等功能函数；获取栈顶两个操作数的功能函数，辅助计算函数的实现；以及计算函数，是计算表达式的入口。

1. //模拟计算器
2. **class** Calculator {
3. **public**:
4. //构造函数
5. Calculator() { expre = Queue(); };
6. //析构函数
7. ~Calculator() {};
8. //初始化
9. **void** Initialize();
10. **private**:
11. Queue expre;
12. //模拟栈
13. LinkStack box;
14. //输入算数表达式。
15. **void** Input();
16. **void** Run();
17. //从栈中获取操作数
18. **bool** Get2operands(**int**& left, **int**& right);
19. //根据操作符进行计算
20. **bool** Calculate(Node\*¤t);
21. //清空队列和栈
22. **void** Clear()
23. {
24. **while** (expre.IsEmpty() == 0)
25. {
26. expre.DeQueue();
27. }
28. **while** (box.IsEmpty() == 0)
29. {
30. box.Pop();
31. }
32. }
33. };

**3 功能实现**

**3.1 后缀表达式转换功能的实现**

**3.1.1后缀表达式转换功能流程图**



**3.1.2后缀表达式转换功能核心代码**

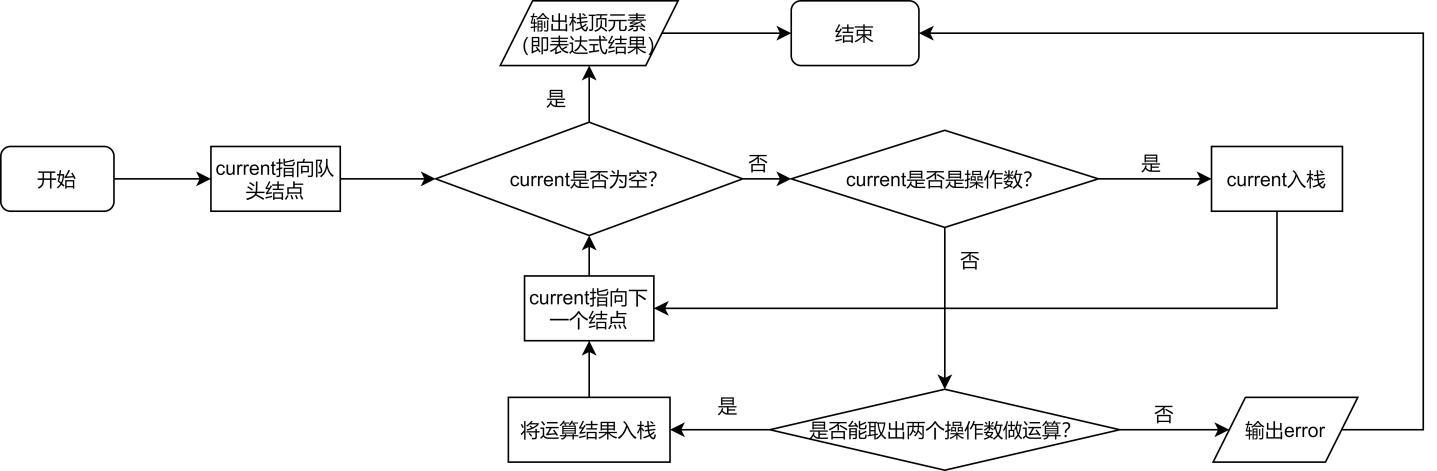
1. //将中缀表达式转换为后缀表达式
2. **void** Queue::Postfix(Node\*current)
3. {
4. LinkStack box;
5. box.Push(current);
6. current = current->next;
7. **while** (!box.IsEmpty() || current->charnum != '#')
8. {
9. //是操作数，直接输出至新队列
10. **if** (current->type == 0)
11. {
12. **this**->EnQueue(current);
13. current = current->next;
14. }
15. //是操作符，进行判断
16. **else** **if** (current->type == 1)
17. {
18. //进栈
19. **if** (current->icp > box.top->isp)
20. {
21. box.Push(current);
22. current = current->next;
23. }
24. //退栈并输出
25. **else** **if** (current->icp < box.top->isp)
26. {
27. **this**->EnQueue(box.top);
28. box.Pop();
29. }
30. //退栈但不输出
31. **else** **if** (current->icp == box.top->isp)
32. {
33. **if** (box.top->charnum == '(')
34. {
35. current = current->next;
36. }
37. box.Pop();
38. }
39. }
40. }
41. }

**3.1.3后缀表达式转换功能截屏示例**



**3.2** 计算后缀表达式功能的实现

**3.2.**1计算后缀表达式功能流程图



**3.2.2计算后缀表达式功能核心代码**

从栈顶退出两个操作数，分别赋值给left变量和right变量，若成功取出两个操作数则返回true，否则返回false。

1. **bool** Calculator::Get2operands(**int**& left, **int**& right)
2. {
3. //取两个操作数
4. **if** (box.IsEmpty() == 0)
5. {
6. right = box.top->intnum;
7. box.Pop();
8. }
9. **else**
10. {
11. **return** **false**;
12. }
13. **if** (box.IsEmpty() == 0)
14. {
15. left = box.top->intnum;
16. box.Pop();
17. }
18. **else**
19. {
20. **return** **false**;
21. }
22. **return** **true**;
23. };

Calculate函数的参数为一个结点类型，里面存储了当前的运算符。先从栈顶获取两个操作数，然后根据当前结点内的运算符做运算，最后将结果储存在一个新的结点变量中，并使其入栈。若操作成功则返回true，否则给出错误提醒：表达式有误！，并返回false。

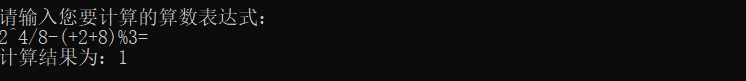
1. //计算后缀表达式
2. **bool** Calculator::Calculate(Node\*& current)
3. {
4. //左右操作数和结果
5. **int** left, right;
6. Node\* temp = **new** Node;
7. //根据操作符进行运算
8. **if** (**this**->Get2operands(left, right) == **true**)
9. {
10. **switch** (current->charnum)
11. {
12. **case**'+':temp->intnum = left + right; box.Push(temp); **break**;
13. **case**'-':temp->intnum = left - right; box.Push(temp); **break**;
14. **case**'\*':temp->intnum = left \* right; box.Push(temp); **break**;
15. **case**'/':temp->intnum = left / right; box.Push(temp); **break**;
16. **case**'%':temp->intnum = left % right; box.Push(temp); **break**;
17. **case**'^':
18. **int** value = left;
19. **for** (**int** i = 0; i < right - 1; i++)
20. {
21. value = value \* left;
22. }
23. temp->intnum = value;
24. box.Push(temp);
25. **break**;
26. }
27. current = current->next;
28. **return** **true**;
29. }
30. **else**
31. {
32. cout << "表达式有误！请重新输入" << endl;
33. **return** **false**;
34. }
35. };

从左至右依次扫描后缀表达式，若当前项为操作数，直接入栈，若当前项为操作符，调用Calculate函数进行计算。当后缀表达式扫描完后，整个表达式也就计算完成了，将结果输出。

1. **void** Calculator:: Run()
2. {
3. **this**->Input();
4. //获得后缀表达式
5. Queue postexpre;
6. postexpre.Postfix(**this**->expre.front);
7. Node\* current = postexpre.front;
8. **while** (current != NULL)
9. {
10. //操作数进栈
11. **if** (current->type == 0)
12. {
13. box.Push(current);
14. current = current->next;
15. }
16. //操作符运算
17. **if** (current->type == 1)
18. {
19. **if** (current->charnum == '#')
20. {
21. current = current->next;
22. **continue**;
23. }
24. **if** (**this**->Calculate(current) == **false**)
25. {
26. **return**;
27. };
28. }
29. }
30. cout << "计算结果为：" << box.top->intnum << endl<<endl;
31. };

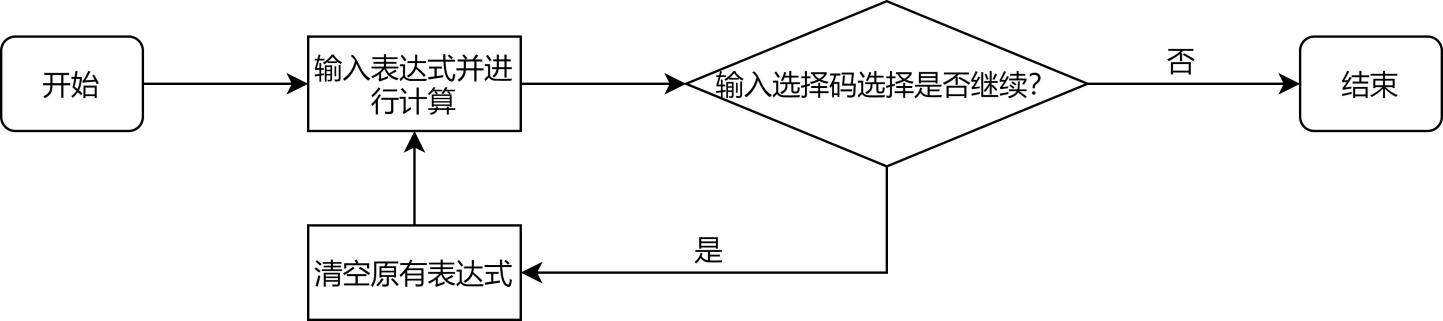
**3.2.3计算后缀表达式功能截屏示例**





**3.3 计算器初始化及运行功能的实现**

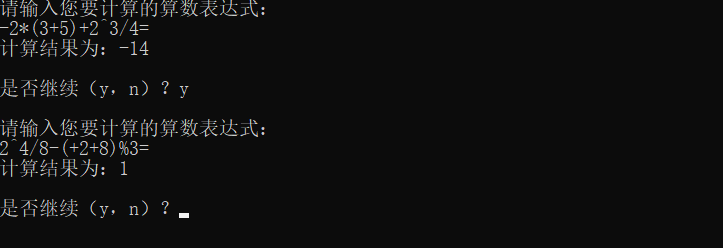
**3.3.1计算器初始化及运行功能流程图**

****

**3.3.2计算器初始化及运行功能核心代码**

1. //初始化
2. **void** Calculator::Initialize()
3. {
4. **this**->Run();
5. **this**->Clear();
6. **char** option = '#'; **int** iscontinue = 1;
7. **while** (iscontinue)
8. {
9. cout << "是否继续（y，n）？";
10. cin >> option;
11. cout << endl;
12. **switch** (option)
13. {
14. **case**'y':**this**->Run(); **this**->Clear(); **break**;
15. **case**'n':iscontinue = 0; **break**;
16. }
17. }
18. }

**3.3.4计算器初始化及运行功能截屏示例**



**4测试**

**4.1计算功能测试**

**测试用例：**-2\*(3+5)+2^3/4=

**预期结果：**-14

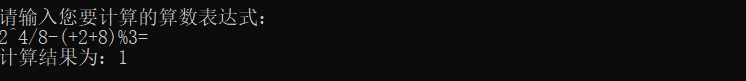
**实验结果：**



**测试用例：**2^4/8-(+2+8)%3=

**预期结果：**1

**实验结果：**



**测试用例：**(3+4)/(2%5)-100+3\*33=

**预期结果：**2

**实验结果：**



**4.2单目运算符计算功能测试**

**测试用例：**-2+78=

**预期结果：**76

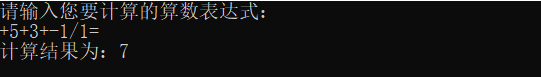
**实验结果：**



**测试用例：**+5+3+-1/1=

**预期结果：**7

**实验结果：**



**4.3表达式错误提醒功能测试**

**测试用例：**2+5/0=

**预期结果：**除零错误！请重新输入

**实验结果：**



**测试用例：**2+++3/1+2\*2=

**预期结果：**表达式有误！请重新输入

**实验结果：**

