算数表达式求解

作 者 姓 名： 王凌

学 号： 1951504

指 导 教 师： 张颖 \_\_

学院、 专业： 软件工程 \_\_

同济大学

Tongji University

目录

[1 分析 1](#_Toc51)

[1.1 项目要求 1](#_Toc30156)

[1.2 功能分析 1](#_Toc31860)

[2 设计 2](#_Toc5203)

[2.1 Stack类设计 2](#_Toc25249)

[2.2 isValid()函数设计 2](#_Toc10666)

[2.3 compute函数设计 3](#_Toc9122)

[2.4 calculation()函数设计 3](#_Toc22961)

[2.5 主函数设计 4](#_Toc22540)

[3 实现 4](#_Toc5655)

[3.1 Stack类实现 4](#_Toc8340)

[3.1.1 Stack()构造函数实现 4](#_Toc9074)

[3.1.2 push()压栈函数实现 4](#_Toc9032)

[3.1.3 pop()出栈函数实现 4](#_Toc31780)

[3.1.4 top()，empty()函数的实现 5](#_Toc21351)

[3.1.5 isFull()，clear()，size()函数的实现 5](#_Toc10059)

[3.2 isValid()函数实现 6](#_Toc24839)

[3.2.1 括号的匹配问题，空括号问题 6](#_Toc3151)

[3.2.2 含有无法处理的字符 7](#_Toc22256)

[3.2.3单目表达式+-(正负)合理性的判断 7](#_Toc20432)

[3.2.4双目表达式合理性的判断 7](#_Toc24663)

[3.2.5等号位置合理性的判断 8](#_Toc27898)

[3.3 compute()函数的实现 8](#_Toc15707)

[3.5 calculation()函数的实现 9](#_Toc5773)

[3.5.1函数设计部分摘录+部分解释 9](#_Toc25708)

[3.5.2 函数代码 10](#_Toc1308)

[3.5 主函数实现 12](#_Toc17)

[4 测试 13](#_Toc17490)

[4.1 功能测试 13](#_Toc13216)

[4.1.1 测试1 13](#_Toc24375)

[4.1.2 测试2 13](#_Toc9952)

[4.1.3 测试3 14](#_Toc3022)

[4.1.4 测试4 15](#_Toc6072)

[4.1.4 测试5 15](#_Toc14474)

[4.2 边界测试 16](#_Toc27222)

[4.2.1括号不匹配，存在空括号 16](#_Toc12895)

[4.2.2非法字符 17](#_Toc13161)

[4.2.3单目运算符合理性错误 17](#_Toc4231)

[4.2.4双目运算符合理性错误 18](#_Toc21869)

[4.2.5等号合理性错误 18](#_Toc15145)

# 1 分析

## 1.1 项目要求

从键盘上输入中缀算数表达式，包括括号，计算出表达式的值。

项目要求：

1.程序对所有输入的表达式作简单的判断，如表达式有错，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

2.能处理单目运算符：+或-。

## 1.2 功能分析

项目的要求主要有两个部分。

首先是对所有输入的表达式都做判断，如果表达式有错，要给出相关的提示。在这一部分我认为最重要的是括号匹配的问题，因此我首先用栈写出了括号匹配，栈里面存的是左括号的位置，所以不仅能判断括号是否匹配，还能输出是哪个括号不匹配。在此之外，还有几个点要简要判断一下：含有本程序中无法处理的字符（尤其是由于本程序用一个字节来存储的，无法处理中文输入法下的括号）；出现了空括号；正负号之后没有数字或者括号；有不合理的+、—、\*、/、^、%的情况（用排除法，非合理的情况即为不合理）；表达式之中出现了=，或者没有以=结尾。

第二部分就是求值。求中缀表达式我所知道的有两种办法，一种是转化为可以被计算机识别的后缀表达式（又叫“逆波兰表达式”）；另一种是不改变中缀表达式的形式，直接用两个栈（数字栈和符号栈），根据定义的运算优先级来计算最终结果。我经过课内的学习，并在网络上查阅相关资料之后，决定用第二种方法，也就是不转化直接计算。

# 2 设计

## 2.1 Stack类设计

包括两个私有成员变量list数组和栈顶位置top0。

公有成员函数如下：

Stack();//构造函数，初始化栈

void push(const T &item);//将元素item压入栈

T pop();//将栈顶元素弹出栈

void clear();//将栈清空

const T &top() const;//访问栈顶元素

bool empty() const;//测试是否栈满

bool isFull() const;//测试是否栈空

int size() const;//输出栈的大小

## 2.2 isValid()函数设计

在这一部分我认为最重要的是括号匹配的问题和单目运算符号正负的合理性判断，除此之外还有几个小点需要注意一下。

1.我首先用栈写出了括号匹配，栈里面存的是左括号的位置，所以不仅能判断括号是否匹配，还能输出具体是哪个括号不匹配；

2.含有本程序中无法处理的字符（尤其是由于本程序用一个字节来存储的，无法处理中文输入法下的括号）；

3.出现了空括号；

4.正负号之后没有数字或者括号，这种情况下是无法表示“正负”，无法单目运算；

5.有不合理的+、—、\*、/、^、%的情况（用排除法，非合理的情况即为不合理）；

6.表达式之中出现了=，或者没有以=结尾。

## 2.3 compute函数设计

这是一个简单的函数，外面输入符号以及两个数字，用switch判断符号的类型，并且最终返回结算的值就可以了。

## 2.4 calculation()函数设计

这是本程序之中最重要最关键的部分。

首先定义栈内运算顺序，用map将字符和数字联系起来，数字越大，优先级越高，栈内(定义为1，+-定义为2，\*/定义为3，%定义为3，^定义为5。（这一部分我当时找资料没找全，所以做复杂了，因为括号在栈内与在栈外的优先级是不同的，所以应该定义两个优先级，但我只定义了一个，所以后面有很多关于括号的判断，但幸运的是最终运行结果都是正确的）

同时定义两个栈num\_stack存循环到当前位置栈内的数字，sig存循环到当前位置栈内的字符。用一个循环从0位置一直读到中缀序列长度减一的位置，即忽略=，因为我的程序没有定义=号的优先度。

有四种情况下做压栈工作：

1. 循环中缀序列读的是数字，那么将字符数字转化为整型数字然后压到num\_stack栈之中；
2. 遇到“(”直接压进sig运算符号栈。；
3. 如果符号栈为空，遇到符号之后直接压栈；
4. 循环到的字符优先级大于前者，则继续压栈；

有三种情况下做退栈、计算操作：

1. 遇到“）”直接退栈并计算到栈顶元素为“(”，最后链栈顶元素为“(”也要出栈；
2. 遇到“=”那么说明剩下的没有括号了，直接按照符号优先级进行计算。
3. 优先级小于等于前者,必将弹出之前所有大于等于他优先级的运算符,再加入这个运算符

## 2.5 主函数设计

要用一个while循环，通过输入y,n来判断程序是否继续进行。每次清空数字栈num\_stack之后，清空输入流防止重复读string类型的时候出现吞首个字符的情况。当输入表达式的时候，前中后都不能出现空格，输入完先判断表达式是否合法，如果合法，则调用calculation函数进行计算。

# 3 实现

## 3.1 Stack类实现

### 3.1.1 Stack()构造函数实现

栈的自定义构造函数，栈顶初始化为-1。

1. **template** <**class** T,**int** SIZE>
2. Stack<T,SIZE>::Stack():top0(-1){}//构造函数，栈顶初始化为-1

### 3.1.2 push()压栈函数实现

如果栈满了，则报错；否则将新元素压入栈顶

1. **template** <**class** T,**int** SIZE>
2. **void** Stack<T,SIZE>::push(**const** T &item)//将元素item压入栈
3. {
4. list[++top0]=item;//将新元素压入栈顶
5. }

### 3.1.3 pop()出栈函数实现

如果栈为空，则报错；否则返回栈顶元素，并将其弹出

1. **template** <**class** T,**int** SIZE>
2. T Stack<T,SIZE>::pop()//将栈顶元素弹出栈
3. {
4. **return** list[top0--];//返回栈顶元素，并将其弹出栈顶
5. }

### 3.1.4 top()，empty()函数的实现

访问栈顶元素，若为空则报错。

测试栈是否为空，判断栈顶指针是否为-1。

1. **template** <**class** T,**int** SIZE>
2. **const** T &Stack<T,SIZE>::top() **const**//访问栈顶元素
3. {
4. assert(!empty());//如果栈为空，则报错
5. **return** list[top0];//返回栈顶元素
6. }
7. **template** <**class** T,**int** SIZE>
8. **bool** Stack<T,SIZE>::empty() **const**//测试栈是否空
9. {
10. **return** top0==-1;
11. }

### 3.1.5 isFull()，clear()，size()函数的实现

测试栈是否为满，判断栈顶指针是否等于SIZE-1；

清空栈则是让栈顶指针指向-1。

1. **template** <**class** T,**int** SIZE>
2. **bool** Stack<T,SIZE>::isFull() **const**//测试是否栈满
3. {
4. **return** top0==SIZE-1;
5. }
6. **template** <**class** T,**int** SIZE>
7. **void** Stack<T,SIZE>::clear()//清空栈
8. {
9. top0=-1;
10. }
11. **template** <**class** T,**int** SIZE>
12. **int** Stack<T,SIZE>::size() **const**//输出栈的大小
13. {
14. **return** top0;
15. }

## 3.2 isValid()函数实现

### 3.2.1 括号的匹配问题，空括号问题

用栈写出了括号匹配，栈里面存的是左括号的位置，循环从左到右遍历，遇到左括号就将左括号的序号入栈，每次遇到一个右括号就就判断，如果栈不为空，则左括号栈弹出一个；如果栈为空，说明这个位置的右括号没有左括号向匹配，输出“没有与第i个位置的右括号匹配的左括号！”。做完之后，如果左括号栈还不为空，说明还有左括号没有右括号向匹配，用while循环弹出栈，并输出“没有与第s0.top()个位置的左括号匹配的右括号！”。

1. //首先是括号匹配问题，需要用栈来记录括号的位置
2. //如果不匹配将输出括号不匹配处的位置
3. **int** N=expr.length();
4. Stack<**int**> s0;
5. **for**(**int** i=1;i<N;i++)//在表达式之中搜索左右括号
6. {
7. **if**(expr[i-1]=='(') s0.push(i);//左括号位置进栈
8. **if**(expr[i-1]==')')
9. {
10. **if**(s0.empty()==**false**) s0.pop();//栈不为空的时候
11. **else**// if(s0.empty())
12. {
13. cout<<"没有与第"<<i<<"个位置的右括号匹配的左括号！"<<endl;
14. **return** 0;
15. }
16. }
17. }
18. **while**(!s0.empty())//栈中还有左括号
19. {
20. cout<<"没有与第"<<s0.top()<<"个位置的左括号匹配的右括号！"<<endl;
21. s0.pop();
22. **return** 0;
23. }

出现了空括号()：

1. **if**(expr[i]=='(' && i+1<N && expr[i+1]==')')
2. //判断是否出现了连续的左右括号
3. {
4. cout<<"出现了空括号()"<<endl;
5. **return** 0;
6. }

### 3.2.2 含有无法处理的字符

含有本程序中无法处理的字符（尤其是由于本程序用一个字节来存储的，无法处理中文输入法下的括号）

1. **if**(expr[i]!='+'&&expr[i]!='-'&&expr[i]!='\*'&&expr[i]!='/'&&expr[i]!='%'&&expr[i]!='^'&&expr[i]!='('&&expr[i]!=')'&&expr[i]!='='&&!(expr[i]<='9'&&expr[i]>='0')&&expr[i]!='.')
2. {
3. cout<<"表达式含有非法字符，注意必须在英文输入法下输入括号"<<endl;
4. **return** 0;
5. }

### 3.2.3单目表达式+-(正负)合理性的判断

正负号之后没有数字或者括号，这种情况下是无法表示“正负”，无法单目运算

1. **if**(expr[i]=='+' || expr[i]=='-')
2. **if**(i==0 || expr[i-1]=='(')
3. {
4. **if**(!(i+1<N && ((expr[i+1]=='(') || (expr[i+1]<='9' && expr[i+1]>='0'))))
5. {
6. cout<<"+-后面没有数字或者("<<endl;
7. **return** 0;
8. }
9. }
10. //只有"i+1<N,并且左边是)或者数字,并且右边是(或者数字"这种情况是合理的，排除这种情况之后，其它情况是不合理的

### 3.2.4双目表达式合理性的判断

有不合理的+、—、\*、/、^、%的情况（用排除法，非合理的情况即为不合理）

1. **else** **if**(!(i+1<N && (expr[i-1]==')' || (expr[i-1]<='9'&&expr[i-1]>='0')) && ((expr[i+1]=='(')||(expr[i+1]<='9'&&expr[i+1]>='0'))))
2. {
3. cout<<"不合理的+-"<<endl;
4. **return** 0;
5. }
6. **if**(expr[i]=='^' || expr[i]=='%' || expr[i]=='\*' || expr[i]=='/')
7. //只有"i-1>=0,i+1<N,并且左边是)或者数字,并且右边是(或者数字"这种情况是合理的，排除这种情况之后，其它情况是不合理的
8. **if**(!(i-1>=0 && i+1<N && ((expr[i-1]==')' || (expr[i-1]<='9'&&expr[i-1]>='0'))&& ((expr[i+1]=='(')||(expr[i+1]<='9'&&expr[i+1]>='0')))))
9. {
10. cout<<"不合理的\*/^%"<<endl;
11. **return** 0;
12. }

### 3.2.5等号位置合理性的判断

表达式之中出现了=，或者没有以=结尾

1. **if**(i==N-1 && expr[i]!='=')//判断最后一位是否是=
2. {
3. cout<<"不以=结尾"<<endl;
4. **return** 0;
5. }
6. **if**(i!=N-1 && expr[i]=='=')//判断除了最后一位还有没有其它位置出现=
7. {
8. cout<<"表达式中出现了="<<endl;
9. **return** 0;
10. }

## 3.3 compute()函数的实现

这是一个简单的函数，外面输入符号以及两个数字，用switch判断符号的类型，并且最终返回结算的值就可以了。

1. **int** compute(**char** c,**int** a,**int** b)
2. {
3. **switch**(c)
4. {
5. **case** '+':**return**(a+b);**break**;
6. **case** '-':**return**(a-b);**break**;
7. **case** '\*':**return**(a\*b);**break**;
8. **case** '/':**return**(a/b);**break**;
9. **case** '^':**return**(pow(a,b));**break**;
10. **case** '%':**return**(((**int**)a)%((**int**)b));**break**;
11. **default**:**return** 0;
12. }
13. }

## 3.5 calculation()函数的实现

### 3.5.1函数设计部分摘录+部分解释

这是本程序之中最重要最关键的部分。

首先定义栈内运算顺序，用map将字符和数字联系起来，数字越大，优先级越高，栈内(定义为1，+-定义为2，\*/定义为3，%定义为3，^定义为5。（这一部分我当时找资料没找全，所以做复杂了，因为括号在栈内与在栈外的优先级是不同的，所以应该定义两个优先级，但我只定义了一个，所以后面有很多关于括号的判断，但幸运的是最终运行结果都是正确的）

同时定义两个栈num\_stack存循环到当前位置栈内的数字，sig存循环到当前位置栈内的字符。用一个循环从0位置一直读到中缀序列长度减一的位置，即忽略=，因为我的程序没有定义=号的优先度。

有四种情况下做压栈工作：

1. 循环中缀序列读的是数字，那么将字符数字转化为整型数字然后压到num\_stack栈之中；
2. 遇到“(”直接压进sig运算符号栈。；
3. 如果符号栈为空，遇到符号之后直接压栈；
4. 循环到的字符优先级大于前者，则继续压栈；

有三种情况下做退栈、计算操作：

1. 遇到“）”直接退栈并计算到栈顶元素为“(”，最后链栈顶元素为“(”也要出栈；
2. 遇到“=”那么说明剩下的没有括号了，直接按照符号优先级进行计算。
3. 优先级小于等于前者,必将弹出之前所有大于等于他优先级的运算符,再加入这个运算符。

补充：由于括号在栈内与栈外的优先级不同，我的代码之中有相当一部分冗余结构，例如用while循环进行无括号的计算这一个部分，我在三种做退栈、计算操作的步骤中都用到了相似的代码。这显得代码不是那么巧夺天工，耦合性不强，但是综合来看，由于是分情况考虑的，程序的鲁棒性还是较强的。

### 3.5.2 函数代码

1. Stack<**int**> num\_stack;//数字栈
2. **int** calculation(**const** string &expr)
3. {
4. map<**char**,**int**> priority;//存储运算符的优先级
5. priority['(']=1;
6. priority['+']=2;priority['-']=2;
7. priority['\*']=3;priority['/']=3;
8. priority['%']=4;priority['^']=5;
9. Stack<**char**> sig;//存储运算符
10. **for**(**int** i=0;i<expr.length()-1;i++)//不读空格
11. {
12. //处理单目运算符
13. **bool** f0=0;//记录数字的正负，等于0的时候为正
14. **if**((expr[i]=='+' || expr[i]=='-') && (i == 0 || expr[i - 1]=='('))//当+-被当做正负来用时
15. **if**(expr[i]!='+')//当-为符号时
16. {
17. i++;//这个符号不压栈
18. f0=1;//记录这个数为负数（记录f0=1）
19. }
20. **else** i++;
21. **if**(expr[i]<='9' && expr[i]>='0')//把字符串转换为数字
22. {
23. **int** pos=i+1;//初始化，pos指向数字的第一位
24. **while**(1)
25. {
26. **if**(pos<expr.length() && expr[pos]<='9' && expr[pos]>='0') pos++;
27. **else** **break**;
28. }
29. **int** number=0;
30. number=atoi(expr.substr(i,pos-i).c\_str());//atoi是将字符串转化为整型数字
31. **if**(f0) number\*=(-1);//如果-为负号，则让数字乘以负一
32. num\_stack.push(number);
33. i=pos-1;//更新已经操作到的字符数组的位置
34. }
35. **else** **if**(expr[i]=='(') sig.push(expr[i]);//左括号直接压入
36. **else** **if**(expr[i]!=')')
37. {
38. **if**(sig.empty()) sig.push(expr[i]);//栈为空直接入栈
39. **else**
40. {
41. //与栈顶符号比较优先级
42. **int** topStackPriority=priority[sig.top()];
43. **if**(priority[expr[i]]>topStackPriority) sig.push(expr[i]);
44. **else**
45. {
46. //优先级小于等于前者,必将弹出之前所有大于等于他优先级的运算符,再加入这个运算符
47. **while**(!sig.empty() && priority[expr[i]]<=topStackPriority)
48. {
49. //a左,b右
50. **int** b=num\_stack.top();
51. num\_stack.pop();
52. **int** a=num\_stack.top();
53. num\_stack.pop();
54. num\_stack.push(compute(sig.top(),a,b));
55. sig.pop();
56. topStackPriority=priority[sig.top()];
57. }
58. sig.push(expr[i]);
59. }
61. }
62. }
63. **else** **if**(expr[i]==')')//等于右括号的时候一直做到左括号处
64. {
65. **int** topStackPriority=priority[sig.top()];
66. **while**(!sig.empty() && priority[expr[i]]<=topStackPriority && sig.top()!='(')
67. {
68. topStackPriority=priority[sig.top()];
69. //a左,b右
70. **int** b=num\_stack.top();
71. num\_stack.pop();
72. **int** a=num\_stack.top();
73. num\_stack.pop();
74. num\_stack.push(compute(sig.top(),a,b));
75. sig.pop();
76. }
77. sig.pop();//把(弹出去
78. }
79. }
80. //当做到最后还有一个=号的时候，还需要把剩余没做完的做完
81. **int** topStackPriority=priority[sig.top()];
82. **while**(!sig.empty() && priority['=']<=topStackPriority)
83. {
84. **if**(sig.top()!='=')
85. {
86. topStackPriority=priority[sig.top()];
87. //a左,b右
88. **int** b=num\_stack.top();
89. num\_stack.pop();
90. **int** a=num\_stack.top();
91. num\_stack.pop();
92. num\_stack.push(compute(sig.top(),a,b));
93. }
94. sig.pop();
95. }
96. **return** num\_stack.top();//最后剩下的值就是所求的值
97. }

## 3.5 主函数实现

要用一个while循环，通过输入y,n来判断程序是否继续进行。每次清空数字栈num\_stack之后，清空输入流防止重复读string类型的时候出现吞首个字符的情况。当输入表达式的时候，前中后都不能出现空格，输入完先判断表达式是否合法，如果合法，则调用calculation函数进行计算。

1. **int** main()
2. {
3. **while**(**true**)
4. {
5. num\_stack.clear();
6. cout<<"输入表达式：(表达式中间,包括=符号之后都不能出现空格)"<<endl;
7. string str="\n";
8. cin.clear(); cin.sync();
9. getline(cin,str);
10. **if**(isValid(str))
11. cout<<calculation(str)<<endl;
12. cout<<"是否继续（y，n）？"<<endl;
13. **char** isContinue;
14. cin>>isContinue;
15. **while**(isContinue!='y' && isContinue!='n')
16. {
17. cout<<"没有这个选项，请重新输入（y，n）："<<endl;
18. cin>>isContinue;
19. }
20. **if**(isContinue=='n') **break**;
21. }
22. **return** 0;
23. }

# 4 测试

## 4.1 功能测试

### 4.1.1 测试1

（行末尾不能有空格！）

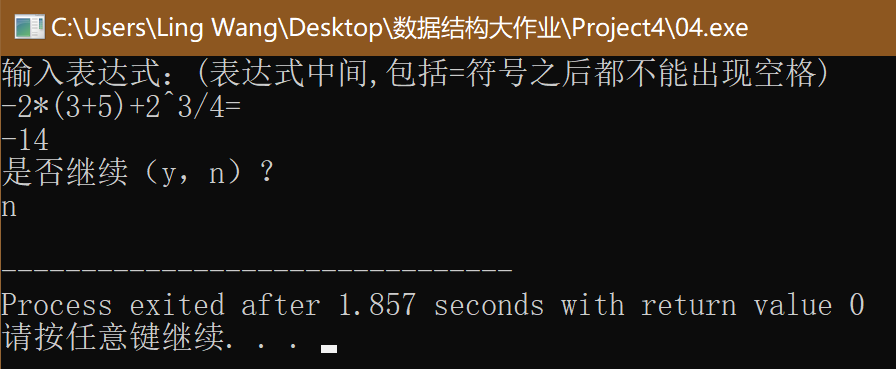
**测试用例**：

-2\*(3+5)+2^3/4=

n

**实验结果：**

应该是-14



### 4.1.2 测试2

**测试用例：**

-2\*(3+5)+2^3/4=

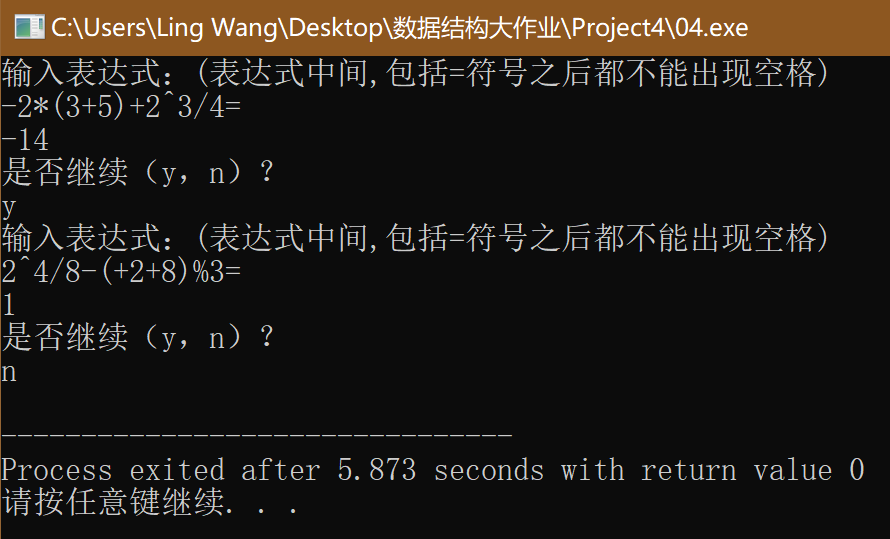
y

2^4/8-(+2+8)%3=

n

**实验结果：**

应该是-14和1



### 4.1.3 测试3

**测试用例：**

10+(-7\*(-9))=

n

**实验结果：**



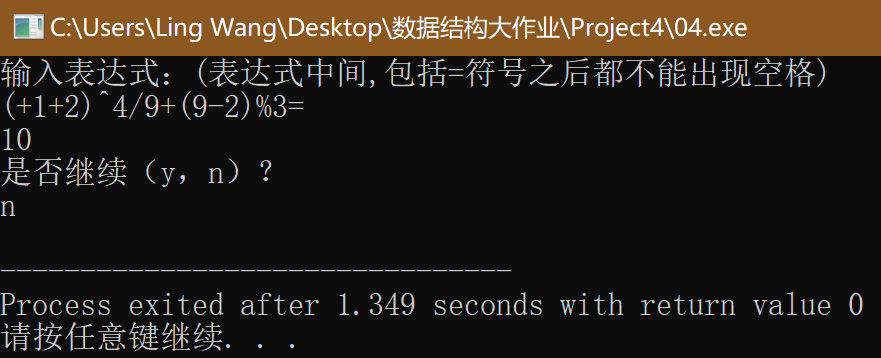
### 4.1.4 测试4

**测试用例：**

(+1+2)^4/9+(9-2)%3=

n

**实验结果：**



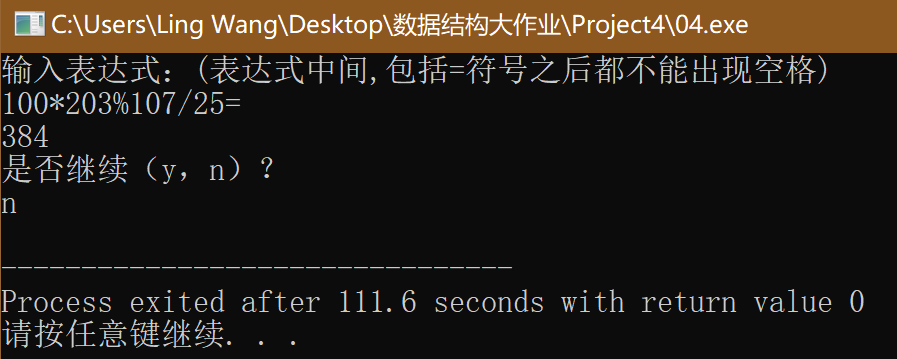
### 4.1.4 测试5

**测试用例：**

100\*203%107/25=

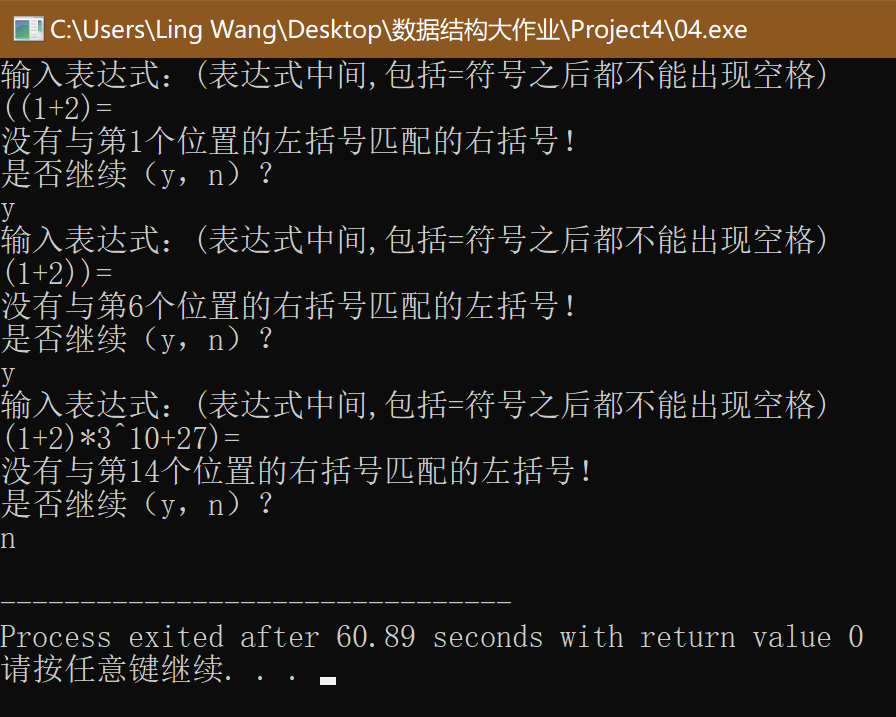
**实验结果：**

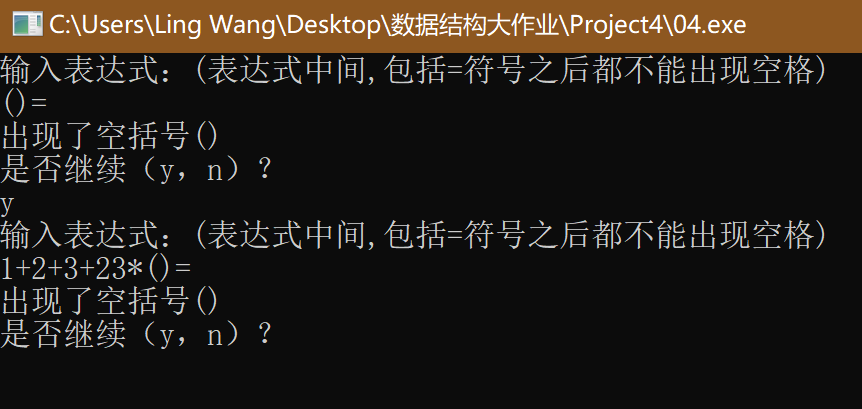
定义的先进行取模运算！



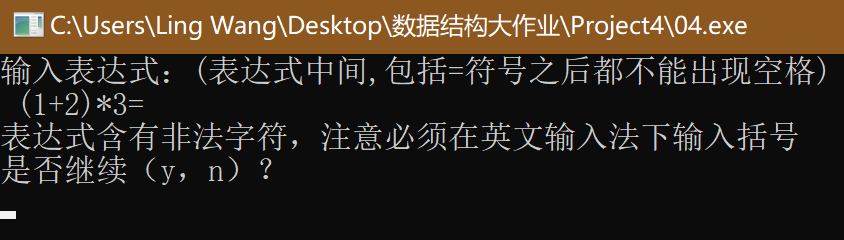
## 4.2 边界测试

### 4.2.1括号不匹配，存在空括号

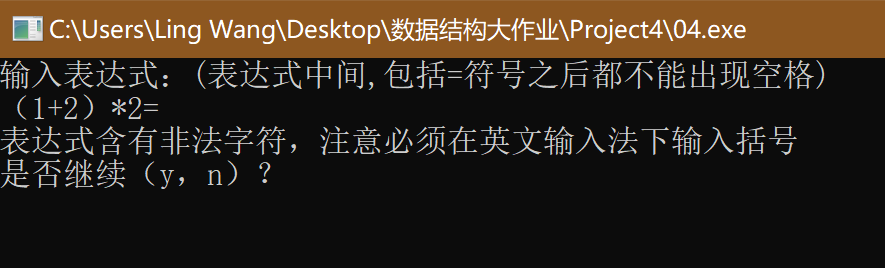




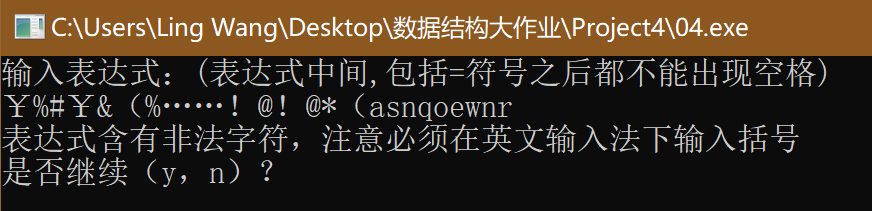
### 4.2.2非法字符



式子左边出现了空格

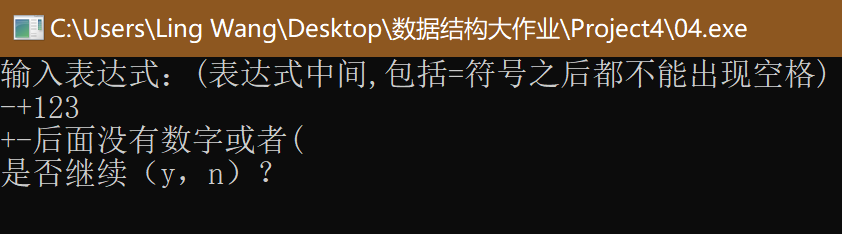


使用了中文输入法下的括号

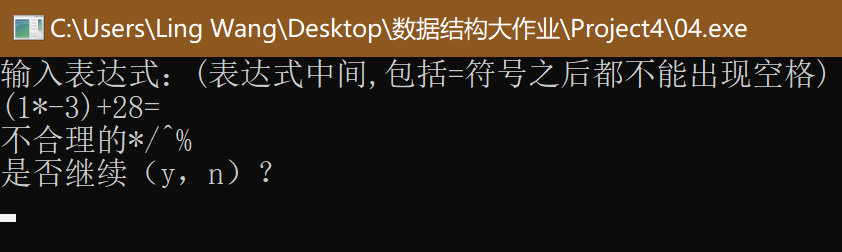


其它乱敲的违法字符

### 4.2.3单目运算符合理性错误



### 4.2.4双目运算符合理性错误



### 4.2.5等号合理性错误

