中缀表达式实验报告

一、题目再现：

中缀表达式是我们熟悉的表达式形式。为了能正确表示运算的先后顺序，中缀表达式中难免要出现括号。假设我们的表达式中只允许有圆括号。

读入一个浮点数为操作数的中缀表达式后，对该表达式进行运算。

要求中缀表达式以一个字符串的形式读入，可含有加、减、乘、除运算符和左、右括号，并假设该表达式以“#”作为输入结束符。

如输入“3.5*(20+4)-1#”，则程序运行结果应为 83。

要求可单步显示输入序列和栈的变化过程。并考虑算法的健壮性，当表达式错误时，要给出错误原因的提示。

二、解题思路：

当我们遇到一个符号时，是否立即执行它取决于下一个符号和它的优先度。

如果将括号和终止符也看成符号，我们可以得到如下的优先度表

	+	-	*	/	(	)	#
+	>	>	<	<	<	>	>
-	>	>	<	<	<	>	>
*	>	>	>	>	<	>	>
/	>	>	>	>	<	>	>
(	<	<	<	<	<	=	
)	>	>	>	>		>	>
#	<	<	<	<	<		=

当读到的当前运算符优先级低于前一个时，可以计算前一个运算符——因此必须记录前一个运算符，即有一个读入运算符的逆序（栈的特点）

计算某个运算符时，它所对应的操作数应该是最后两个操作数——因此必须记录操作数的逆序（栈的特点）

优先度与操作数无关，分别处理，独立储存，都使用栈储存

算法：

设置两个栈：符号栈和数字栈

将终止标记符压入符号栈，从左往右读取字符串

假设读入到的符号为 ch，取出栈顶 ch1，比较 ch 和 ch1 的优先度

ch=ch1:

ch 是括号：

去除括号,符号栈进行出栈操作，读入下一个符号 ch

ch 是#终止符：

输出栈顶，程序结束

ch>ch1:

将 ch 压入符号栈，读入下一个符号 ch

ch<ch1:

先计算表达式中前一个子式

符号栈弹出栈顶 ch1

数字栈弹出两次 y x

其中第一次出栈的 y 是右操作数，第二次出栈的 x 是左操作数；
进行运算：res=x-ch1-y 并将结果压入数字栈

三、设计思路：

1、存储数字和符号时

在逻辑层面使用栈结构，在物理层面选择大数组，编写较为方便。

2、主要使用两个函数

(1) solve()：用来从左往右扫描，读取字符串中的数字和符号

(2) work()：用于 solve 读取到字符时，将它与栈中字符比较，并处理

3、考虑到代码健壮性

(1) 采取整行读入，删除所有不合法字符（包括空格）

(2) 在第一位的负数，采取负号前添 0 使其合法

(3) 遇到错误时，使用 string_error()函数和错误识别码，输出可能的错误原因

4、get_mark(char c)函数有两个作用

(1) 判断是否为合法符号(+-*/#)，若不合法返回 0，用于将字符分段

(2) 若为合法符号，则返回符号在优先度表中的下标，用于比较当前符号与栈顶符号

5、get_num_in_str(string s,int x,int y)函数：

来读取两个符号间的数字，根据小数点来定位整数和小数部分

6、出现错误时：

(1) 用 enum 定义一种枚举类型：Error_code，用于在栈中返回错误类型（注：error_code 是 std 中的保留词）

(2) 当调用栈出现错误时，根据返回的 Error_code 调用 string_error(int i)函数，传递相应的错误识别码 i，将错误进一步转化成用户可以看懂的语言并输出，然后直接结束整个程序。

7、将功能性函数放进 utility.h 中，同时 Stack.h 和 utility.h 中不能使用 using namespace std, 防止套太多时函数产生歧义

8、使用泛型 template，Stack 可以根据定义随机改变栈的类型

9、使用 switch 替代 if 来判断符号，提升效率

10、当栈操作成功时，则按要求输出本次操作

四、时空复杂度分析：

1、时间复杂度

(1) 健壮性处理：O(n)

(2) 提取字符串中的数字：O(n)

(3) 从左往右进行栈处理：

最坏情况：每个符号最多只会被处理两遍（进栈，出栈）O(2*n)

最好情况：左边符号优先级总是大于右边符号优先级 O(n)

总结：

时间复杂度为 O(n)级别，有常数，但一般不会超过 O(10*n)

2、空间复杂度

数字栈最大长度为表达式中所有数字个数

符号栈最大长度为表达式中所有符号个数

总结：

空间复杂度也为 O(n)级别

五、调试用例：(1) -3*(-2+1.1*2)/2-(-3)-(5.2+2.5)#　　带负号，较为复杂的式子

```
this is a program solving Infix Expression
you can input a Math Expression,ending with a '#'
such as :3*(2+1.1*2.)/2-3-(5.2+2.5)#
     or :-3*(-2+1.1*2)/2-(-3)-(5.2+2.5)#
     or :5aa.aa1aa+aa2aa.aa5aa#a#aa##
please input a math expression:-3*(-2+1.1*2)/2-(-3)-(5.2+2.5)
after formatting:0-3*(0-2+1.1*2)/2-(0-3)-(5.2+2.5)
mark   stack:push '#'
number stack:push 0
# < -
mark   stack:push '-'
number stack:push 3
- < *
mark   stack:push '*'
* < (
mark   stack:push '('
number stack:push 0
( < -
mark   stack:push '-'
number stack:push 2
- > +
mark   stack:pop '-'
number stack:pop 2
number stack:pop 0
0 - 2 = -2
number stack:push -2
( < +
mark   stack:push '+'
number stack:push 1.1
+ < *
mark   stack:push '*'
number stack:push 2
* > )
mark   stack:pop '*'
number stack:pop 2
number stack:pop 1.1
1.1 * 2 = 2.2
number stack:push 2.2
+ > )
mark   stack:pop '+'
number stack:pop 2.2
number stack:pop -2
-2 + 2.2 = 0.2
number stack:push 0.2
( = )
mark   stack:pop '('
* > /
mark   stack:pop '*'
number stack:pop 0.2
number stack:pop 3
3 * 0.2 = 0.6
number stack:push 0.6
- < /
mark   stack:push '/'
number stack:push 2
/ > -
mark   stack:pop '/'
number stack:pop 2
number stack:pop 0.6
0.6 / 2 = 0.3
number stack:push 0.3
- > -
mark   stack:pop '-'
number stack:pop 0.3
number stack:pop 0
0 - 0.3 = -0.3
number stack:push -0.3
# < -
mark   stack:push '-'
- < (
mark   stack:push '('
number stack:push 5.2
( < +
mark   stack:push '+'
number stack:push 2.5
+ > )
mark   stack:pop '+'
number stack:pop 2.5
number stack:pop 5.2
5.2 + 2.5 = 7.7
number stack:push 7.7
( = )
mark   stack:pop '('
- > #
mark   stack:pop '-'
number stack:pop 7.7
number stack:pop 2.7
2.7 - 7.7 = -5
number stack:push -5
# = #
answer is:-5
```

(2) 5+5　　缺少#

```
this is a program solving Infix Expression
you can input a Math Expression,ending with a '#'
such as :3*(2+1.1*2.)/2-3-(5.2+2.5)#
     or :-3*(-2+1.1*2)/2-(-3)-(5.2+2.5)#
     or :5aa.aa1aa+aa2aa.aa5aa#a#aa##
please input a math expression:5+5
there is no '#' in your string,please check it
```

(3) 5++5#　不合法的式子

```
this is a program solving Infix Expression
you can input a Math Expression,ending with a '#'
such as :3*(2+1.1*2.)/2-3-(5.2+2.5)#
     or :-3*(-2+1.1*2)/2-(-3)-(5.2+2.5)#
     or :5aa.aa1aa+aa2aa.aa5aa#a#aa##
please input a math expression:5++5#
after formatting:5++5#
mark   stack:push '#'
number stack:push 5
# < +
mark   stack:push '+'
+ > +
mark   stack:pop '+'
number stack:pop 5
error!the string is incorrect
the string may lack numbers,please check it
```

(4) #5+5　　#号前没有数字

```
this is a program solving Infix Expression
you can input a Math Expression,ending with a '#'
such as :3*(2+1.1*2.)/2-3-(5.2+2.5)#
     or :-3*(-2+1.1*2)/2-(-3)-(5.2+2.5)#
     or :5aa.aa1aa+aa2aa.aa5aa#a#aa##
please input a math expression:#5+5#
after formatting:#5+5#
mark   stack:push '#'
# = #
error!the string is incorrect
there may be no number before '#',please check it
```

(5) 5 aa. aa1aa +a a2aa. aa5a a#a#aa##

中间带空格和其余不产生歧义的字符也可进行计算

```
this is a program solving Infix Expression
you can input a Math Expression,ending with a '#'
such as :3*(2+1.1*2.)/2-3-(5.2+2.5)#
     or :-3*(-2+1.1*2)/2-(-3)-(5.2+2.5)#
     or :5aa.aa1aa+aa2aa.aa5aa#a#aa##
please input a math expression:5 aa. aa1aa +a a2aa. aa5a a#a#aa##
after formatting:5.1+2.5####
mark   stack:push '#'
number stack:push 5.1
# < +
mark   stack:push '+'
number stack:push 2.5
+ > #
mark   stack:pop '+'
number stack:pop 2.5
number stack:pop 5.1
5.1 + 2.5 = 7.6
number stack:push 7.6
# = #
answer is:7.6
```

(6) (5.+6.#　　符号搭配不当，缺半个括号

```
this is a program solving Infix Expression
you can input a Math Expression,ending with a '#'
such as :3*(2+1.1*2.)/2-3-(5.2+2.5)#
     or :-3*(-2+1.1*2)/2-(-3)-(5.2+2.5)#
     or :5aa.aa1aa+aa2aa.aa5aa#a#aa##
please input a math expression:(5.+6.#
after formatting:(5.+6.#
mark   stack:push '#'
# < (
mark   stack:push '('
number stack:push 5
( < +
mark   stack:push '+'
number stack:push 6
+ > #
mark   stack:pop '+'
number stack:pop 6
number stack:pop 5
5 + 6 = 11
number stack:push 11
(   #
error!the string is incorrect
improper mark matchong,please check it
```

六、后缀表达式求值与括号匹配

1、 后缀表达式求值：

与中缀表达式相似，但比中缀表达式简单得多，只需要一个栈来存储数字即可

打包在 test2_Postfix_expression

2、 括号匹配

从左往右读取字符串，建立一个栈存储扫描到的三种左括号，读取到右括号时，判断栈顶与它是否匹配即可，最后再判断栈是否为空（左括号是否被匹配完）

打包在 test2_brackets

七、总结

本次实验是第一次使用自己编写的结构，异常处理折腾了很久，一开始只是使用简单的 cout，效果并不好，并且结构很臃肿，后来改成了 throw，throw 在调用时很不友好，每次使用函数都要 catch 一次错误，太麻烦，后来改成了自定义枚举类型 Error_code，在使用和编写时比较友好。

本次实验提升了对于栈结构的理解和运用的熟练度。

对于写较为大型的程序有了一定的了解。

对于程序的规则有了更深层次的认识，并能从用户角度考虑问题。

代码能力得到了很大的提升。

对于物理结构、逻辑结构和算法的理解更深了一步。