**一、实验题目**

用栈实现计算器

**二、实验目的**

1、掌握栈的基本操作：插入、删除、查找等运算。

2、掌握栈的存储特点及其实现。

**三、实验设备与环境**

微型计算机、macOS操作系统 、pyqt6、QT designer、Pycharm系列软件

**四、实验内容**

利用栈实现一个科学计算器。

**五、概要设计（思路、算法、步骤等）**

**5.1 思路**

1. 首先利用 Qt Designer 设计出计算器界面。
2. 利用 PyUIC 将.ui 文件生成为.py 文件，再编写一个MyWindow类去继承这个类，在 MyWindow 类中添加按钮响应、鼠标单击监听、键盘事件响应等。
3. 最后在等号这个按钮写入计算的函数。

**5.2 步骤**

算法分为三个步骤：

1. 判断框中的表达式是否合法：（judge\_leg()函数）

①判断括号是否配对，利用栈数据结构。

②将表达式中的所有数字、操作符分离，并加入列表之中。分离步骤使用列表操作。

③分别判断左右括号两边的数字，操作符情况，例如左括号的右边是操作符，左边是数字。

④判断除括号外的其他操作符是否多次出现，或者第一位是否为除号或乘号，最后一位是否为非（）外的操作符。

2.将合法的中缀表达式转换为后缀表达式：（to\_postfix()函数）

①将遍历列表，如遇到操作数则直接输出。

②若遇到左括号，则直接压入栈内。

③若遇到右括号，则不断弹出栈顶元素，知道遇到左括号。

④若遇到运算符，则与栈顶元素相比较，若优先级高于栈顶元素，则压入栈。若优先级低于或等于栈顶元素，则将栈顶元运算符弹出并输出，然后比较新的栈顶运算符，直到优先级大于栈顶元素，或者栈空，再将改运算符入栈。

⑤若列表处理完毕，则按顺序输出栈中的所有运算符。

3.计算后缀表达式：（cal()函数与caltrans ()函数）

①遍历列表，若遇到操作数，就压入栈中。

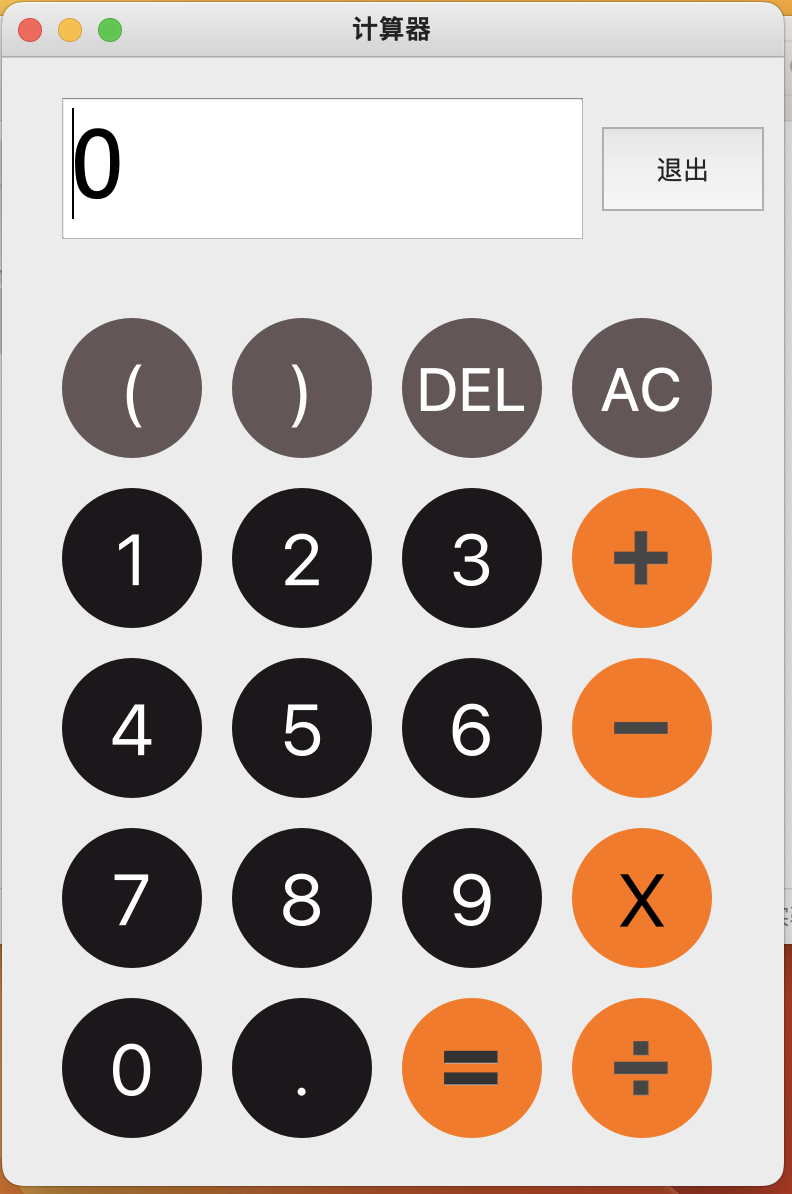
②若遇到操作符，则弹出两个元素，并进行计算，最后压入栈中。其中caltrans() 函数用来计算两个操作数一个操作符的计算结果。

③最后栈顶元素即为计算所得值。

**六、详细设计（核心代码、算法流程图等）**

UI界面：

*# Form implementation generated from reading ui file 'stack.ui'  
#  
# Created by: PyQt6 UI code generator 6.1.0  
#  
# WARNING: Any manual changes made to this file will be lost when pyuic6 is  
# run again. Do not edit this file unless you know what you are doing.*from PyQt6 import QtCore, QtGui, QtWidgets  
  
  
class Ui\_Dialog(object):  
 def setupUi(self, Dialog):  
 Dialog.setObjectName("Dialog")  
 Dialog.resize(391, 564)  
 self.add = QtWidgets.QPushButton(Dialog)  
 self.add.setGeometry(QtCore.QRect(285, 215, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.add.setFont(font)  
 self.add.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.add.setStyleSheet("QPushButton{\n"  
"background-color:rgb(240,123,44);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.add.setObjectName("add")  
 self.sub = QtWidgets.QPushButton(Dialog)  
 self.sub.setGeometry(QtCore.QRect(285, 300, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.sub.setFont(font)  
 self.sub.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.sub.setStyleSheet("QPushButton{\n"  
"background-color:rgb(240,123,44);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.sub.setObjectName("sub")  
 self.mul = QtWidgets.QPushButton(Dialog)  
 self.mul.setGeometry(QtCore.QRect(285, 385, 70, 70))  
 font = QtGui.QFont()  
 font.setFamily("Verdana")  
 font.setPointSize(36)  
 self.mul.setFont(font)  
 self.mul.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.mul.setStyleSheet("QPushButton{\n"  
"background-color:rgb(240,123,44);\n"  
"color:black;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}\n"  
"")  
 self.mul.setObjectName("mul")  
 self.div = QtWidgets.QPushButton(Dialog)  
 self.div.setGeometry(QtCore.QRect(285, 470, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.div.setFont(font)  
 self.div.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.div.setStyleSheet("QPushButton{\n"  
"background-color:rgb(240,123,44);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.div.setObjectName("div")  
 self.equal = QtWidgets.QPushButton(Dialog)  
 self.equal.setGeometry(QtCore.QRect(200, 470, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.equal.setFont(font)  
 self.equal.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.equal.setStyleSheet("QPushButton{\n"  
"background-color:rgb(240,123,44);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.equal.setObjectName("equal")  
 self.rightbrack = QtWidgets.QPushButton(Dialog)  
 self.rightbrack.setGeometry(QtCore.QRect(115, 130, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.rightbrack.setFont(font)  
 self.rightbrack.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.rightbrack.setStyleSheet("QPushButton{\n"  
"background-color:rgb(98,86,86);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}\n"  
"")  
 self.rightbrack.setObjectName("rightbrack")  
 self.dele = QtWidgets.QPushButton(Dialog)  
 self.dele.setGeometry(QtCore.QRect(200, 130, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(30)  
 self.dele.setFont(font)  
 self.dele.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.dele.setStyleSheet("QPushButton{\n"  
"background-color:rgb(98,86,86);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}\n"  
"")  
 self.dele.setObjectName("dele")  
 self.leftbrack = QtWidgets.QPushButton(Dialog)  
 self.leftbrack.setGeometry(QtCore.QRect(30, 130, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.leftbrack.setFont(font)  
 self.leftbrack.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.leftbrack.setStyleSheet("QPushButton{\n"  
"background-color:rgb(98,86,86);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}\n"  
"\n"  
"")  
 self.leftbrack.setObjectName("leftbrack")  
 self.ac = QtWidgets.QPushButton(Dialog)  
 self.ac.setGeometry(QtCore.QRect(285, 130, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(30)  
 self.ac.setFont(font)  
 self.ac.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.ac.setStyleSheet("QPushButton{\n"  
"background-color:rgb(98,86,86);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}\n"  
"")  
 self.ac.setObjectName("ac")  
 self.dot = QtWidgets.QPushButton(Dialog)  
 self.dot.setGeometry(QtCore.QRect(115, 470, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.dot.setFont(font)  
 self.dot.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}")  
 self.dot.setObjectName("dot")  
 self.btn0 = QtWidgets.QPushButton(Dialog)  
 self.btn0.setGeometry(QtCore.QRect(30, 470, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.btn0.setFont(font)  
 self.btn0.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}")  
 self.btn0.setObjectName("btn0")  
 self.btn7 = QtWidgets.QPushButton(Dialog)  
 self.btn7.setGeometry(QtCore.QRect(30, 385, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.btn7.setFont(font)  
 self.btn7.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.btn7.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.btn7.setObjectName("btn7")  
 self.btn8 = QtWidgets.QPushButton(Dialog)  
 self.btn8.setGeometry(QtCore.QRect(115, 385, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.btn8.setFont(font)  
 self.btn8.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.btn8.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.btn8.setObjectName("btn8")  
 self.btn9 = QtWidgets.QPushButton(Dialog)  
 self.btn9.setGeometry(QtCore.QRect(200, 385, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.btn9.setFont(font)  
 self.btn9.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.btn9.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.btn9.setObjectName("btn9")  
 self.btn4 = QtWidgets.QPushButton(Dialog)  
 self.btn4.setGeometry(QtCore.QRect(30, 300, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.btn4.setFont(font)  
 self.btn4.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.btn4.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.btn4.setObjectName("btn4")  
 self.btn6 = QtWidgets.QPushButton(Dialog)  
 self.btn6.setGeometry(QtCore.QRect(200, 300, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.btn6.setFont(font)  
 self.btn6.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.btn6.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.btn6.setObjectName("btn6")  
 self.btn5 = QtWidgets.QPushButton(Dialog)  
 self.btn5.setGeometry(QtCore.QRect(115, 300, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.btn5.setFont(font)  
 self.btn5.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.btn5.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.btn5.setObjectName("btn5")  
 self.btn1 = QtWidgets.QPushButton(Dialog)  
 self.btn1.setGeometry(QtCore.QRect(30, 215, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.btn1.setFont(font)  
 self.btn1.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.btn1.setMouseTracking(True)  
 self.btn1.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.btn1.setObjectName("btn1")  
 self.btn3 = QtWidgets.QPushButton(Dialog)  
 self.btn3.setGeometry(QtCore.QRect(200, 215, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.btn3.setFont(font)  
 self.btn3.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.btn3.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.btn3.setObjectName("btn3")  
 self.btn2 = QtWidgets.QPushButton(Dialog)  
 self.btn2.setGeometry(QtCore.QRect(115, 215, 70, 70))  
 font = QtGui.QFont()  
 font.setPointSize(36)  
 self.btn2.setFont(font)  
 self.btn2.setCursor(QtGui.QCursor(QtCore.Qt.CursorShape.PointingHandCursor))  
 self.btn2.setStyleSheet("QPushButton{\n"  
"background-color:rgb(28,23,26);\n"  
"color:white;\n"  
"border-radius:35px;\n"  
"}\n"  
"QPushButton:hover{/\*鼠标悬停\*/\n"  
"background-color:rgb(122,105,97);\n"  
"}\n"  
"QPushButton:pressed{/\*鼠标按下\*/\n"  
"background-color:rgb(122,105,97);\n"  
"border:2px black;\n"  
"border-style:outset;\n"  
"}")  
 self.btn2.setObjectName("btn2")  
 self.display = QtWidgets.QTextEdit(Dialog)  
 self.display.setGeometry(QtCore.QRect(30, 20, 261, 71))  
 font = QtGui.QFont()  
 font.setFamily("Trebuchet MS")  
 font.setPointSize(48)  
 font.setItalic(False)  
 self.display.setFont(font)  
 self.display.setObjectName("display")  
 self.exit = QtWidgets.QPushButton(Dialog)  
 self.exit.setGeometry(QtCore.QRect(300, 30, 81, 51))  
 self.exit.setCheckable(False)  
 self.exit.setObjectName("exit")  
  
 self.retranslateUi(Dialog)  
 QtCore.QMetaObject.connectSlotsByName(Dialog)  
  
 def retranslateUi(self, Dialog):  
 \_translate = QtCore.QCoreApplication.translate  
 Dialog.setWindowTitle(\_translate("Dialog", "计算器"))  
 self.add.setText(\_translate("Dialog", "➕"))  
 self.sub.setText(\_translate("Dialog", "➖"))  
 self.mul.setText(\_translate("Dialog", "X"))  
 self.div.setText(\_translate("Dialog", "➗"))  
 self.equal.setText(\_translate("Dialog", "🟰"))  
 self.rightbrack.setText(\_translate("Dialog", ")"))  
 self.dele.setText(\_translate("Dialog", "DEL"))  
 self.leftbrack.setText(\_translate("Dialog", "("))  
 self.ac.setText(\_translate("Dialog", "AC"))  
 self.dot.setText(\_translate("Dialog", "."))  
 self.btn0.setText(\_translate("Dialog", "0"))  
 self.btn7.setText(\_translate("Dialog", "7"))  
 self.btn8.setText(\_translate("Dialog", "8"))  
 self.btn9.setText(\_translate("Dialog", "9"))  
 self.btn4.setText(\_translate("Dialog", "4"))  
 self.btn6.setText(\_translate("Dialog", "6"))  
 self.btn5.setText(\_translate("Dialog", "5"))  
 self.btn1.setText(\_translate("Dialog", "1"))  
 self.btn3.setText(\_translate("Dialog", "3"))  
 self.btn2.setText(\_translate("Dialog", "2"))  
 self.display.setHtml(\_translate("Dialog", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"  
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"  
"p, li { white-space: pre-wrap; }\n"  
"</style></head><body style=\" font-family:\'Trebuchet MS\'; font-size:48pt; font-weight:400; font-style:normal;\">\n"  
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">0</p></body></html>"))  
 self.exit.setText(\_translate("Dialog", "退出"))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import sys  
 app = QtWidgets.QApplication(sys.argv)  
 Dialog = QtWidgets.QDialog()  
 ui = Ui\_Dialog()  
 ui.setupUi(Dialog)  
 Dialog.show()  
 sys.exit(app.exec())



发现光标在0的左侧。

于是，在class MyWindow(QWidget, Ui\_Dialog)的\_\_init\_\_函数中，插入以下代码：

cursor = self.display.textCursor()  
cursor.movePosition(QTextCursor.MoveOperation.End)  
self.display.setTextCursor(cursor)

核心代码：

栈：

class Stack:  
 def \_\_init\_\_(self):  
 self.\_\_list = []  
  
 def is\_empty(self):  
 return self.\_\_list == []  
  
 def push(self, item):  
 self.\_\_list.append(item)  
  
 def pop(self):  
 if self.is\_empty():  
 return  
 else:  
 return self.\_\_list.pop()  
  
 def top(self):  
 if self.is\_empty():  
 return  
 else:  
 return self.\_\_list[-1]

这段代码定义了一个名为 Stack（栈）的类，其中包含以下四个方法：

\_\_init\_\_：这是类的构造函数，用于初始化一个空的栈，即创建一个空列表。

is\_empty：用于判断栈是否为空。如果栈中没有任何元素，则返回 True，否则返回 False。

push：这是一个将元素添加到栈顶的方法。它接受一个参数 item，将其添加到栈顶，即列表的末尾。

pop：这是一个从栈顶删除元素的方法。如果栈为空，则返回 None。否则，从栈顶弹出元素并将其返回。

top：这是一个返回栈顶元素但不删除它的方法。如果栈为空，则返回 None。否则，返回列表的最后一个元素，即栈顶元素。

这个类的目的是为了模拟栈这种数据结构，具有后进先出（LIFO）的特性。可以使用这个类来实现各种算法，例如括号匹配、表达式求值等等。

def judge\_leg(string):  
 *# 表达式的合法性  
 # 括号配对* mystack = Stack()  
 for c in string:  
 if c == '(': *#如果有左括号* mystack.push('(') *#把它压入栈* if c == ')': *#如果是右括号* if mystack.is\_empty(): *#如果栈是空的，说明括号不匹配，返回0* return 0  
 else:  
 mystack.pop() *#弹出前面的'('* if not mystack.is\_empty(): *#如果结束了里面还有括号，括号不匹配，返回0* return 0  
 list\_all = []  
 list\_op = ['+', '-', '\*', '/', '(', ')']  
 for i in string:  
 list\_all.append(i)  
  
 i = 0  
 while i <= len(list\_all) - 1:  
 if list\_all[i] not in list\_op and i != len(list\_all) - 1 and list\_all[i + 1] not in list\_op:  
 list\_all[i] = list\_all[i] + list\_all[i + 1]  
 del list\_all[i + 1]  
 i = i - 1  
 i = i + 1  
  
 op = 0  
 num = 0  
  
 for i in range(len(list\_all)):  
 if list\_all[i] == '(' and i != 0: *# 左括号的右边是操作符，左边是数字* if list\_all[i - 1] not in list\_op or list\_all[i + 1] in list\_op:  
 return 0  
 if list\_all[i] == ')' and i != len(list\_all) - 1: *# 右括号的右边是数字，左边是操作符* if list\_all[i - 1] in list\_op or list\_all[i + 1] not in list\_op:  
 return 0  
 *# 将表达式分离数字以及符号,并判断表达式的合理性* list\_all = []  
 list\_op = ['+', '-', '\*', '/', '(', ')']  
 for i in string:  
 list\_all.append(i)  
 list\_op2 = ['+', '-', '\*', '/']  
 i = 0  
 while i <= len(list\_all) - 1: *#把数字合并* if list\_all[i] not in list\_op and i != len(list\_all) - 1 and list\_all[i + 1] not in list\_op:  
 list\_all[i] = list\_all[i] + list\_all[i + 1]  
 del list\_all[i + 1]  
 i = i - 1  
 i = i + 1  
 for i in range(len(list\_all)): *#判断表达式是否非法* if list\_all[i] == '(' and i != 0: *#如果list\_all[i]为左括号且i不等于0的时候（注意：因为前面已经判断了括号是匹配的，所以最后一个元素不可能为左括号，所以不需要有这个条件。* if list\_all[i-1] not in list\_op or list\_all[i+1] in list\_op: *#如果左括号的前面的元素为数字或者后面的数为符号，那就是非法的* return 0  
 if list\_all[i] == ')' and i != len(list\_all) - 1: *#如果list\_all[i]为右括号且i不为最后一个元素的时候（注意：因为前面已经判断了括号是匹配的，所以第一个元素不可能为右括号，所以不需要有这个条件。* if list\_all[i - 1] in list\_op and list\_all[i+1] not in list\_op: *#如果右括号的前面的元素为符号 或者后面的数为数字，那就是非法的* return 0  
 if list\_all[i] in list\_op2 and list\_all[i + 1] in list\_op2 and i != len(list\_all) - 1: *#如果有连续的两个+-/\*，也是非法的* return 0  
 if list\_all[0] == '/' or list\_all[0] == '\*' or list\_all[len(list\_all)-1] in list\_op2: *#如果第一个元素为/或\*，也是非法的* return 0  
 return list\_all

这段代码实现了一个判断算术表达式是否合法的函数 judge\_leg(string)

具体实现的过程是首先使用栈 mystack 对表达式中的括号进行匹配判断，如果括号不匹配则表达式不合法，函数返回 0。然后对表达式进行检查，如果出现连续的两个运算符、以及表达式的开头为乘除运算符或结尾为运算符也是不合法的，函数同样返回 0。最后返回分离数字以及符号后的表达式，如果表达式合法则返回处理后的列表，否则返回 0。

总的来说，这段代码实现了一个对算术表达式进行合法性检查的函数，其中使用了栈这个数据结构来检查括号是否匹配。同时，也对表达式中的各种情况进行了判断，以保证表达式的合法性。

def judeg\_div0():  
 return True  
  
def caltrans(a, b, op):  
 a = float(a)  
 b = float(b)  
 if op == '+':  
 res = a + b  
 if res.is\_integer():  
 res = int(res)  
 return res  
 if op == '-':  
 res = a - b  
 if res.is\_integer():  
 res = int(res)  
 return res  
 if op == '\*':  
 res = a \* b  
 if res.is\_integer():  
 res = int(res)  
 return res  
 if op == '/':  
 if b == 0:  
 return judeg\_div0()  
 res = a / b  
 if res.is\_integer():  
 res = int(res)  
 return res

这段代码定义了函数 judge\_div0与caltrans，它有三个参数： a，b 和 op。 a 和 b 是两个数字， op 是一个字符，表示要进行的数学操作。

在函数的开头， a 和 b 被转换为浮点数。然后，该函数根据 op 的值执行不同的数学操作。如果 op 是 '+'，则将 a 和 b 相加，并将结果存储在变量 res 中。如果 res 是整数，则将其转换为整数，然后将其返回（不然太长一串有点丑）。如果 op 是 '-'，'\*' 或 '/'，则执行相应的数学运算，并采取相同的转换和返回逻辑。若除数为0，judge\_div0返回True。该函数可以用于执行基本的加、减、乘、除四则运算。

def to\_postfix(list\_all):  
 list\_op = ['+', '-', '\*', '/', '(', ')']  
 dict\_order = {'+': 1, '-': 1, '\*': 2, '/': 2, '(': 0}  
 list\_res = []  
 list\_res.append('0')  
 mystack = Stack()  
 for i in list\_all:  
 if i not in list\_op:  
 list\_res.append(i)  
 else:  
 if i == '(':  
 mystack.push(i)  
 elif i == ')':  
 while mystack.top() != '(':  
 list\_res.append(mystack.top())  
 mystack.pop()  
 mystack.pop()  
 else:  
 if mystack.is\_empty():  
 mystack.push(i)  
 else:  
 if dict\_order[i] > dict\_order[mystack.top()]:  
 mystack.push(i)  
 else:  
 while not mystack.is\_empty() and dict\_order[i] <= dict\_order[mystack.top()]:  
 list\_res.append(mystack.top())  
 mystack.pop()  
 mystack.push(i)  
  
 while not mystack.is\_empty():  
 list\_res.append(mystack.top())  
 mystack.pop()  
 return list\_res  
  
  
def cal(list\_res): *# 计算后缀表达式* list\_op = ['+', '-', '\*', '/', ')', '(']  
 mystack = Stack()  
 for c in list\_res:  
 if c not in list\_op:  
 mystack.push(c)  
 else:  
 a = mystack.top()  
 mystack.pop()  
 b = mystack.top()  
 mystack.pop()  
 mystack.push(caltrans(b, a, c))  
  
 return mystack.top()

这段代码是将中缀表达式转换为后缀表达式并计算结果的函数。

首先，定义了一个包含加减乘除和括号的运算符列表和一个包含运算符优先级的字典。然后定义了一个空的列表作为结果存储，和一个栈用来进行中转计算。将0添加到结果列表中，方便后续计算。

遍历中缀表达式中的每一个字符，如果该字符不在运算符列表中，则将该字符添加到结果列表中。否则，判断该字符是左括号还是右括号，如果是左括号，则将该字符压入栈中；如果是右括号，则将栈中元素弹出并添加到结果列表中，直到遇到左括号，将左括号从栈中弹出。

如果是其他运算符，则比较该运算符与栈顶元素的优先级，如果大于栈顶元素，则将该运算符压入栈中；否则将栈中元素弹出并添加到结果列表中，直到栈为空或者该运算符的优先级大于栈顶元素。最后将栈中剩余的元素依次弹出并添加到结果列表中，得到后缀表达式。

然后，定义了一个计算后缀表达式的函数，遍历后缀表达式中的每一个字符，如果该字符不在运算符列表中，则将该字符压入栈中。否则，弹出栈顶的两个元素作为运算数，根据该运算符进行计算，并将结果压入栈中。最后栈中剩下的元素即为计算结果，将其返回。

class MyWindow(QWidget, Ui\_Dialog):  
 signal = pyqtSignal(str)  
  
 def \_\_init\_\_(self, parent=None):  
 super(MyWindow, self).\_\_init\_\_(parent)  
 self.setupUi(self)  
 self.setup\_connections()  
 cursor = self.display.textCursor()  
 cursor.movePosition(QTextCursor.MoveOperation.End)  
 self.display.setTextCursor(cursor)  
  
 def setup\_connections(self):  
 *# 数字按钮* self.btn0.clicked.connect(lambda: self.add\_to\_display('0'))  
 self.btn1.clicked.connect(lambda: self.add\_to\_display('1'))  
 self.btn2.clicked.connect(lambda: self.add\_to\_display('2'))  
 self.btn3.clicked.connect(lambda: self.add\_to\_display('3'))  
 self.btn4.clicked.connect(lambda: self.add\_to\_display('4'))  
 self.btn5.clicked.connect(lambda: self.add\_to\_display('5'))  
 self.btn6.clicked.connect(lambda: self.add\_to\_display('6'))  
 self.btn7.clicked.connect(lambda: self.add\_to\_display('7'))  
 self.btn8.clicked.connect(lambda: self.add\_to\_display('8'))  
 self.btn9.clicked.connect(lambda: self.add\_to\_display('9'))  
 self.dot.clicked.connect(lambda: self.add\_to\_display('.'))  
 *# 操作按钮* self.add.clicked.connect(lambda: self.add\_to\_display('+'))  
 self.sub.clicked.connect(lambda: self.add\_to\_display('-'))  
 self.mul.clicked.connect(lambda: self.add\_to\_display('\*'))  
 self.div.clicked.connect(lambda: self.add\_to\_display('/'))  
 self.leftbrack.clicked.connect(lambda: self.add\_to\_display('('))  
 self.rightbrack.clicked.connect(lambda: self.add\_to\_display(')'))  
 *# 其他按钮* self.dele.clicked.connect(lambda: self.delete\_display())  
 self.ac.clicked.connect(lambda: self.clear\_display())  
 self.equal.clicked.connect(lambda: self.calculate())  
 self.exit.clicked.connect(self.click\_button)  
  
 def add\_to\_display(self, value):  
 if self.display.toPlainText() == '0':  
 self.display.clear()  
 self.display.insertPlainText(value)  
  
 def delete\_display(self):  
 cursor = self.display.textCursor()  
 cursor.deletePreviousChar()  
  
 def clear\_display(self):  
 self.display.setText('0')  
 cursor = self.display.textCursor()  
 cursor.movePosition(QTextCursor.MoveOperation.End)  
 self.display.setTextCursor(cursor)  
  
 def calculate(self):  
 string = self.display.toPlainText()  
 if judge\_leg(string) == 0:  
 self.display.setText('Error！')  
 elif judeg\_div0():  
 self.display.setText('Error！')  
 else:  
 self.display.setText(str(cal(to\_postfix(judge\_leg(string)))))  
 cursor = self.display.textCursor()  
 cursor.movePosition(QTextCursor.MoveOperation.End)  
 self.display.setTextCursor(cursor)  
  
 def click\_button(self):  
 confirm\_exit = QMessageBox.question(  
 self, "提示", "你确定要退出吗😭",  
 QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No, QMessageBox.StandardButton.No)  
 if confirm\_exit == QMessageBox.StandardButton.Yes:  
 QApplication.quit()

if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 win = MyWindow()  
 win.show()  
 sys.exit(app.exec())

这段代码定义了一个名为 MyWindow 的窗口类，该类继承自 QWidget 和 Ui\_Dialog 类。其中， QWidget 是 PyQt 库中的一个基本控件类， Ui\_Dialog 是 Qt Designer 设计的 UI 界面类。

在 MyWindow 类中，首先定义了一个名为 signal 的 PyQt 信号，用于在类的实例中发送一个字符串信号。接着， \_\_init\_\_ 方法用于初始化窗口，其中调用了 setupUi 方法将 Ui\_Dialog 设计的 UI 界面添加到窗口中，然后调用了 setup\_connections 方法设置窗口的信号槽连接，并将光标移动到文本框的末尾。

setup\_connections 方法中，分别对数字按钮、操作按钮、其他按钮进行了**信号槽连接**的设置。其中，add\_to\_display 方法用于将按钮对应的值添加到文本框中， delete\_display 方法用于删除文本框中最后一个字符， clear\_display 方法用于清空文本框内容并将光标移动到末尾。calculate 方法用于将文本框中的字符串转换为后缀表达式并计算结果，如果输入的表达式有误则显示 "Error!"。click\_button 方法用于弹出一个消息框，询问用户是否要退出程序。

在 \_\_main\_\_ 函数中，创建了一个 QApplication 实例和一个 MyWindow 实例，将 MyWindow 显示出来，并执行了应用程序的主循环。

**完整代码：**

**caculator.py**

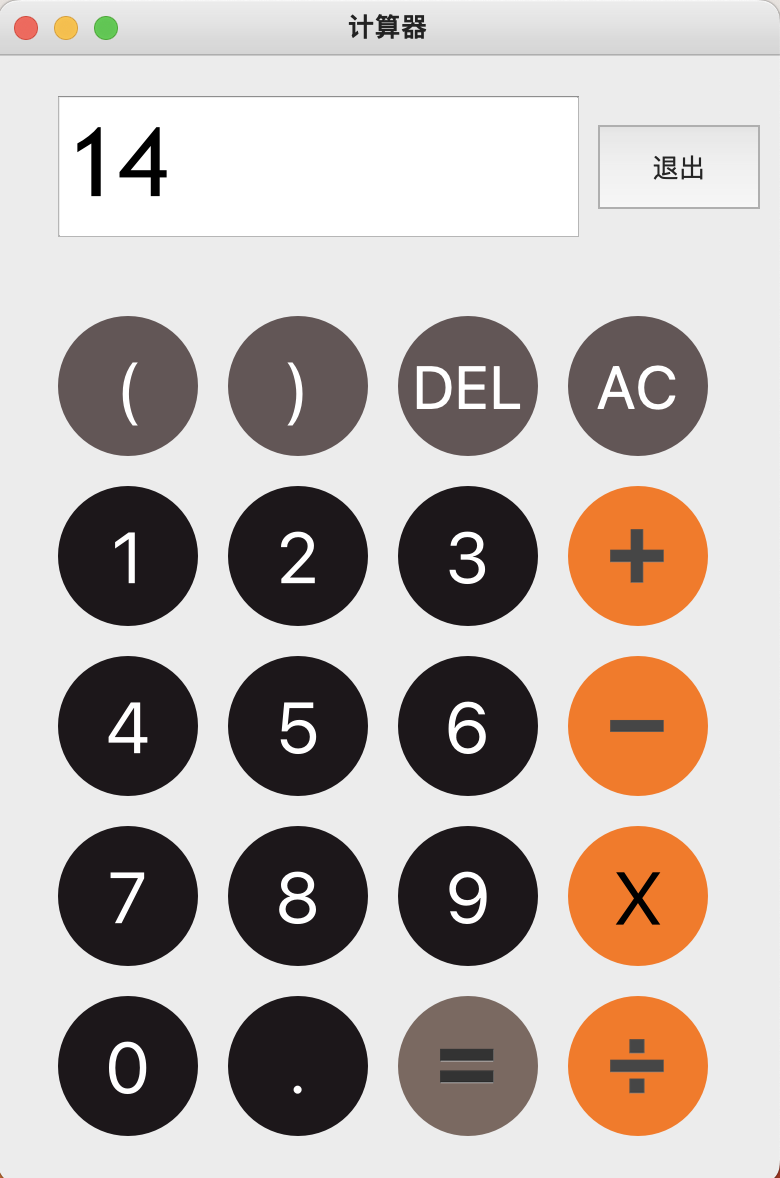
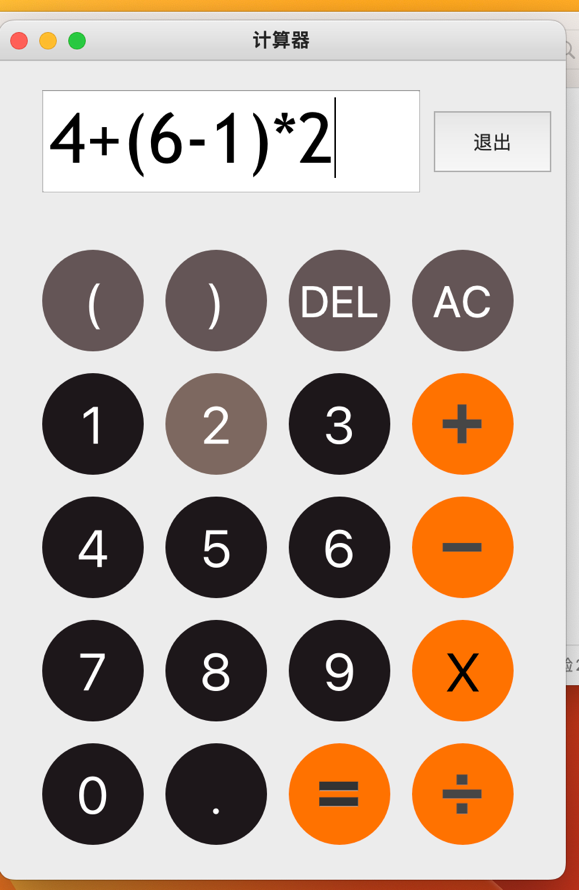
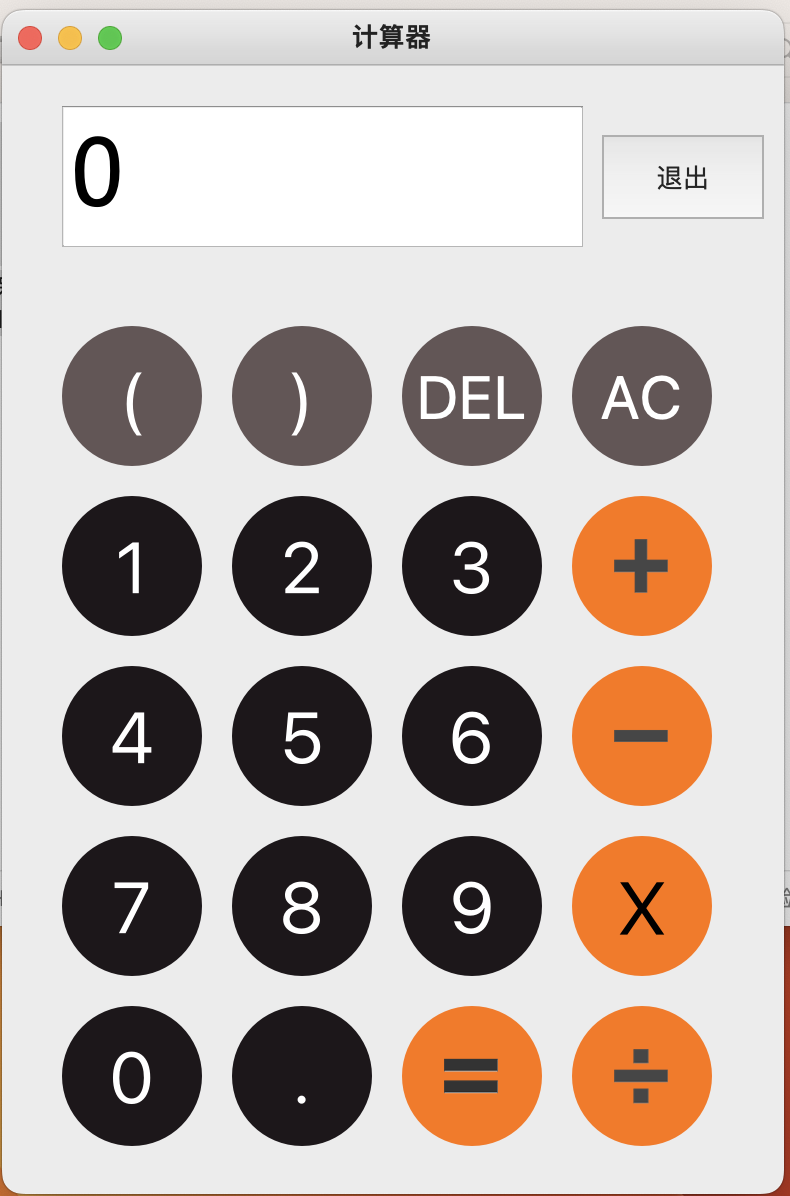
class Stack:  
 def \_\_init\_\_(self):  
 self.\_\_list = []  
  
 def is\_empty(self):  
 return self.\_\_list == []  
  
 def push(self, item):  
 self.\_\_list.append(item)  
  
 def pop(self):  
 if self.is\_empty():  
 return  
 else:  
 return self.\_\_list.pop()  
  
 def top(self):  
 if self.is\_empty():  
 return  
 else:  
 return self.\_\_list[-1]

**main.py**

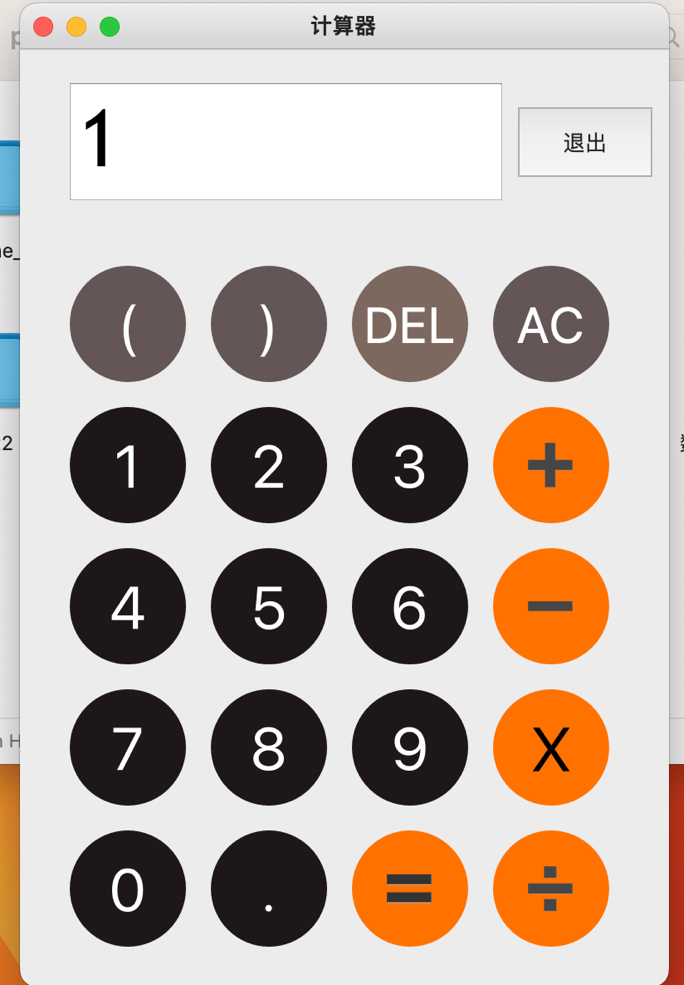
import sys  
  
from PyQt6.QtCore import pyqtSignal  
from PyQt6.QtGui import QTextCursor  
from PyQt6.QtWidgets import QWidget, QApplication, QMessageBox  
from caculator import Stack  
from stack import Ui\_Dialog  
  
  
def judge\_leg(string):  
 *# 表达式的合法性  
 # 括号配对* mystack = Stack()  
 for c in string:  
 if c == '(': *#如果有左括号* mystack.push('(') *#把它压入栈* if c == ')': *#如果是右括号* if mystack.is\_empty(): *#如果栈是空的，说明括号不匹配，返回0* return 0  
 else:  
 mystack.pop() *#弹出前面的'('* if not mystack.is\_empty(): *#如果结束了里面还有括号，括号不匹配，返回0* return 0  
 list\_all = []  
 list\_op = ['+', '-', '\*', '/', '(', ')']  
 for i in string:  
 list\_all.append(i)  
  
 i = 0  
 while i <= len(list\_all) - 1:  
 if list\_all[i] not in list\_op and i != len(list\_all) - 1 and list\_all[i + 1] not in list\_op:  
 list\_all[i] = list\_all[i] + list\_all[i + 1]  
 del list\_all[i + 1]  
 i = i - 1  
 i = i + 1  
  
 op = 0  
 num = 0  
  
 for i in range(len(list\_all)):  
 if list\_all[i] == '(' and i != 0: *# 左括号的右边是操作符，左边是数字* if list\_all[i - 1] not in list\_op or list\_all[i + 1] in list\_op:  
 return 0  
 if list\_all[i] == ')' and i != len(list\_all) - 1: *# 右括号的右边是数字，左边是操作符* if list\_all[i - 1] in list\_op or list\_all[i + 1] not in list\_op:  
 return 0  
 *# 将表达式分离数字以及符号,并判断表达式的合理性* list\_all = []  
 list\_op = ['+', '-', '\*', '/', '(', ')']  
 for i in string:  
 list\_all.append(i)  
 list\_op2 = ['+', '-', '\*', '/']  
 i = 0  
 while i <= len(list\_all) - 1: *#把数字合并* if list\_all[i] not in list\_op and i != len(list\_all) - 1 and list\_all[i + 1] not in list\_op:  
 list\_all[i] = list\_all[i] + list\_all[i + 1]  
 del list\_all[i + 1]  
 i = i - 1  
 i = i + 1  
 for i in range(len(list\_all)): *#判断表达式是否非法* if list\_all[i] == '(' and i != 0: *#如果list\_all[i]为左括号且i不等于0的时候（注意：因为前面已经判断了括号是匹配的，所以最后一个元素不可能为左括号，所以不需要有这个条件。* if list\_all[i-1] not in list\_op or list\_all[i+1] in list\_op: *#如果左括号的前面的元素为数字或者后面的数为符号，那就是非法的* return 0  
 if list\_all[i] == ')' and i != len(list\_all) - 1: *#如果list\_all[i]为右括号且i不为最后一个元素的时候（注意：因为前面已经判断了括号是匹配的，所以第一个元素不可能为右括号，所以不需要有这个条件。* if list\_all[i - 1] in list\_op and list\_all[i+1] not in list\_op: *#如果右括号的前面的元素为符号 或者后面的数为数字，那就是非法的* return 0  
 if list\_all[i] in list\_op2 and list\_all[i + 1] in list\_op2 and i != len(list\_all) - 1: *#如果有连续的两个+-/\*，也是非法的* return 0  
 if list\_all[0] == '/' or list\_all[0] == '\*' or list\_all[len(list\_all)-1] in list\_op2: *#如果第一个元素为/或\*，也是非法的* return 0  
 return list\_all  
  
def judeg\_div0():  
 return True  
  
def caltrans(a, b, op):  
 a = float(a)  
 b = float(b)  
 if op == '+':  
 res = a + b  
 if res.is\_integer():  
 res = int(res)  
 return res  
 if op == '-':  
 res = a - b  
 if res.is\_integer():  
 res = int(res)  
 return res  
 if op == '\*':  
 res = a \* b  
 if res.is\_integer():  
 res = int(res)  
 return res  
 if op == '/':  
 if b == 0:  
 return judeg\_div0()  
 res = a / b  
 if res.is\_integer():  
 res = int(res)  
 return res  
  
  
def to\_postfix(list\_all):  
 list\_op = ['+', '-', '\*', '/', '(', ')']  
 dict\_order = {'+': 1, '-': 1, '\*': 2, '/': 2, '(': 0}  
 list\_res = []  
 list\_res.append('0')  
 mystack = Stack()  
 for i in list\_all:  
 if i not in list\_op:  
 list\_res.append(i)  
 else:  
 if i == '(':  
 mystack.push(i)  
 elif i == ')':  
 while mystack.top() != '(':  
 list\_res.append(mystack.top())  
 mystack.pop()  
 mystack.pop()  
 else:  
 if mystack.is\_empty():  
 mystack.push(i)  
 else:  
 if dict\_order[i] > dict\_order[mystack.top()]:  
 mystack.push(i)  
 else:  
 while not mystack.is\_empty() and dict\_order[i] <= dict\_order[mystack.top()]:  
 list\_res.append(mystack.top())  
 mystack.pop()  
 mystack.push(i)  
  
 while not mystack.is\_empty():  
 list\_res.append(mystack.top())  
 mystack.pop()  
 return list\_res  
  
  
def cal(list\_res): *# 计算后缀表达式* list\_op = ['+', '-', '\*', '/', ')', '(']  
 mystack = Stack()  
 for c in list\_res:  
 if c not in list\_op:  
 mystack.push(c)  
 else:  
 a = mystack.top()  
 mystack.pop()  
 b = mystack.top()  
 mystack.pop()  
 mystack.push(caltrans(b, a, c))  
  
 return mystack.top()  
  
  
class MyWindow(QWidget, Ui\_Dialog):  
 signal = pyqtSignal(str)  
  
 def \_\_init\_\_(self, parent=None):  
 super(MyWindow, self).\_\_init\_\_(parent)  
 self.setupUi(self)  
 self.setup\_connections()  
 cursor = self.display.textCursor()  
 cursor.movePosition(QTextCursor.MoveOperation.End)  
 self.display.setTextCursor(cursor)  
  
 def setup\_connections(self):  
 *# 数字按钮* self.btn0.clicked.connect(lambda: self.add\_to\_display('0'))  
 self.btn1.clicked.connect(lambda: self.add\_to\_display('1'))  
 self.btn2.clicked.connect(lambda: self.add\_to\_display('2'))  
 self.btn3.clicked.connect(lambda: self.add\_to\_display('3'))  
 self.btn4.clicked.connect(lambda: self.add\_to\_display('4'))  
 self.btn5.clicked.connect(lambda: self.add\_to\_display('5'))  
 self.btn6.clicked.connect(lambda: self.add\_to\_display('6'))  
 self.btn7.clicked.connect(lambda: self.add\_to\_display('7'))  
 self.btn8.clicked.connect(lambda: self.add\_to\_display('8'))  
 self.btn9.clicked.connect(lambda: self.add\_to\_display('9'))  
 self.dot.clicked.connect(lambda: self.add\_to\_display('.'))  
 *# 操作按钮* self.add.clicked.connect(lambda: self.add\_to\_display('+'))  
 self.sub.clicked.connect(lambda: self.add\_to\_display('-'))  
 self.mul.clicked.connect(lambda: self.add\_to\_display('\*'))  
 self.div.clicked.connect(lambda: self.add\_to\_display('/'))  
 self.leftbrack.clicked.connect(lambda: self.add\_to\_display('('))  
 self.rightbrack.clicked.connect(lambda: self.add\_to\_display(')'))  
 *# 其他按钮* self.dele.clicked.connect(lambda: self.delete\_display())  
 self.ac.clicked.connect(lambda: self.clear\_display())  
 self.equal.clicked.connect(lambda: self.calculate())  
 self.exit.clicked.connect(self.click\_button)  
  
 def add\_to\_display(self, value):  
 if self.display.toPlainText() == '0':  
 self.display.clear()  
 self.display.insertPlainText(value)  
  
 def delete\_display(self):  
 cursor = self.display.textCursor()  
 cursor.deletePreviousChar()  
  
 def clear\_display(self):  
 self.display.setText('0')  
 cursor = self.display.textCursor()  
 cursor.movePosition(QTextCursor.MoveOperation.End)  
 self.display.setTextCursor(cursor)  
  
 def calculate(self):  
 string = self.display.toPlainText()  
 if judge\_leg(string) == 0:  
 self.display.setText('Error！')  
 elif judeg\_div0():  
 self.display.setText('Error！')  
 else:  
 self.display.setText(str(cal(to\_postfix(judge\_leg(string)))))  
 cursor = self.display.textCursor()  
 cursor.movePosition(QTextCursor.MoveOperation.End)  
 self.display.setTextCursor(cursor)  
  
 def click\_button(self):  
 confirm\_exit = QMessageBox.question(  
 self, "提示", "你确定要退出吗😭",  
 QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No, QMessageBox.StandardButton.No)  
 if confirm\_exit == QMessageBox.StandardButton.Yes:  
 QApplication.quit()  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 win = MyWindow()  
 win.show()  
 sys.exit(app.exec())

**七、测试结果及分析**

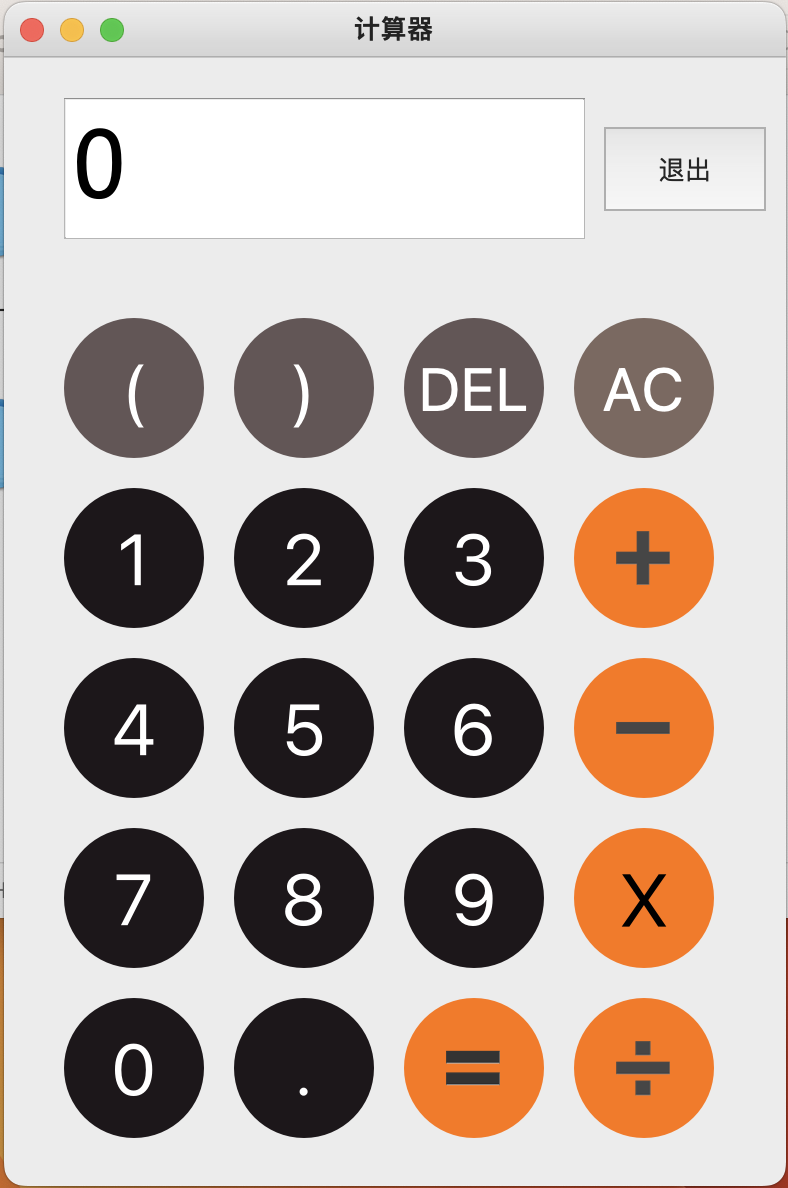
**主页面**



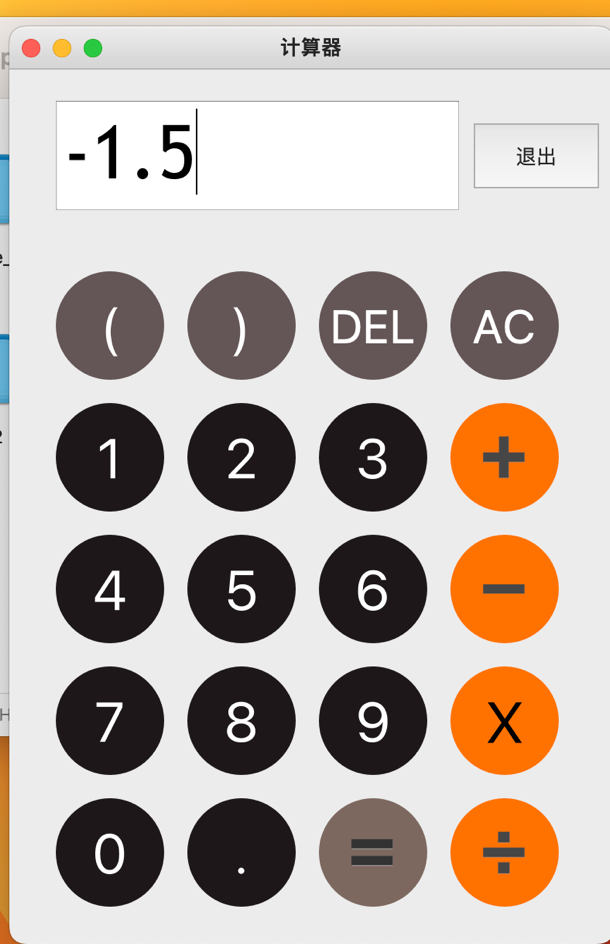
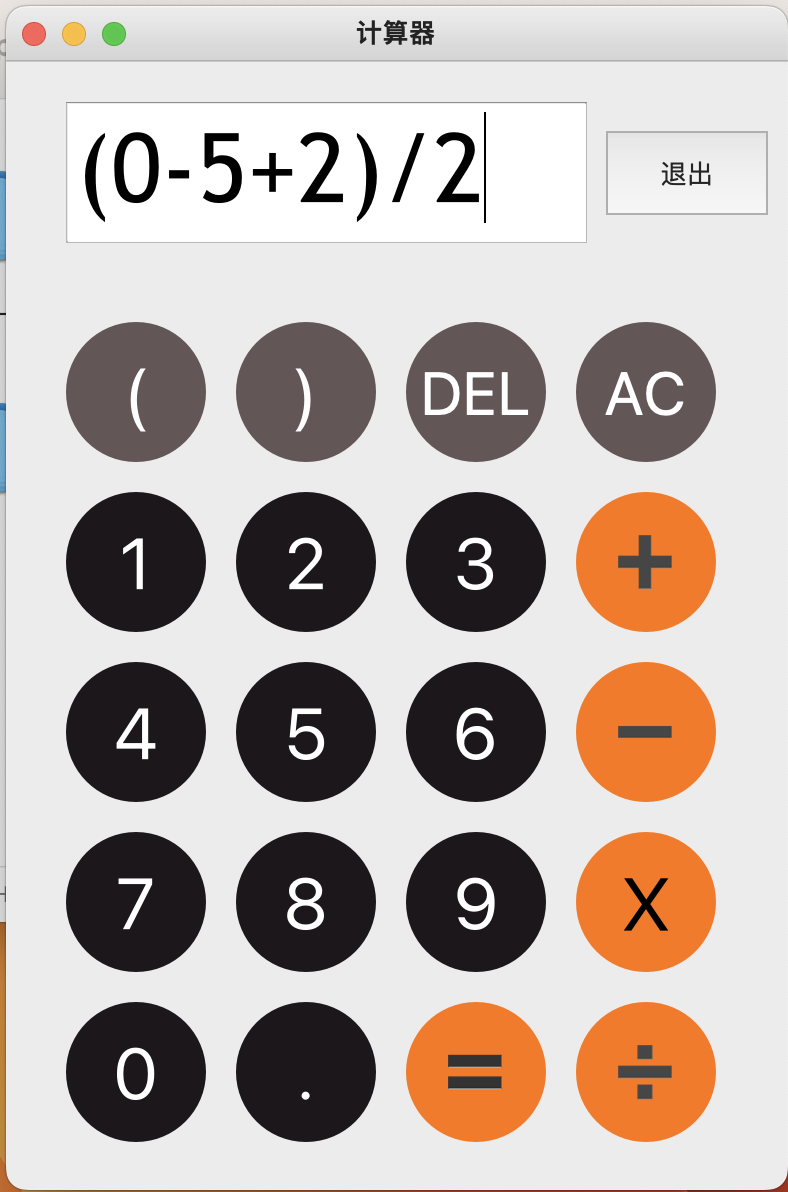
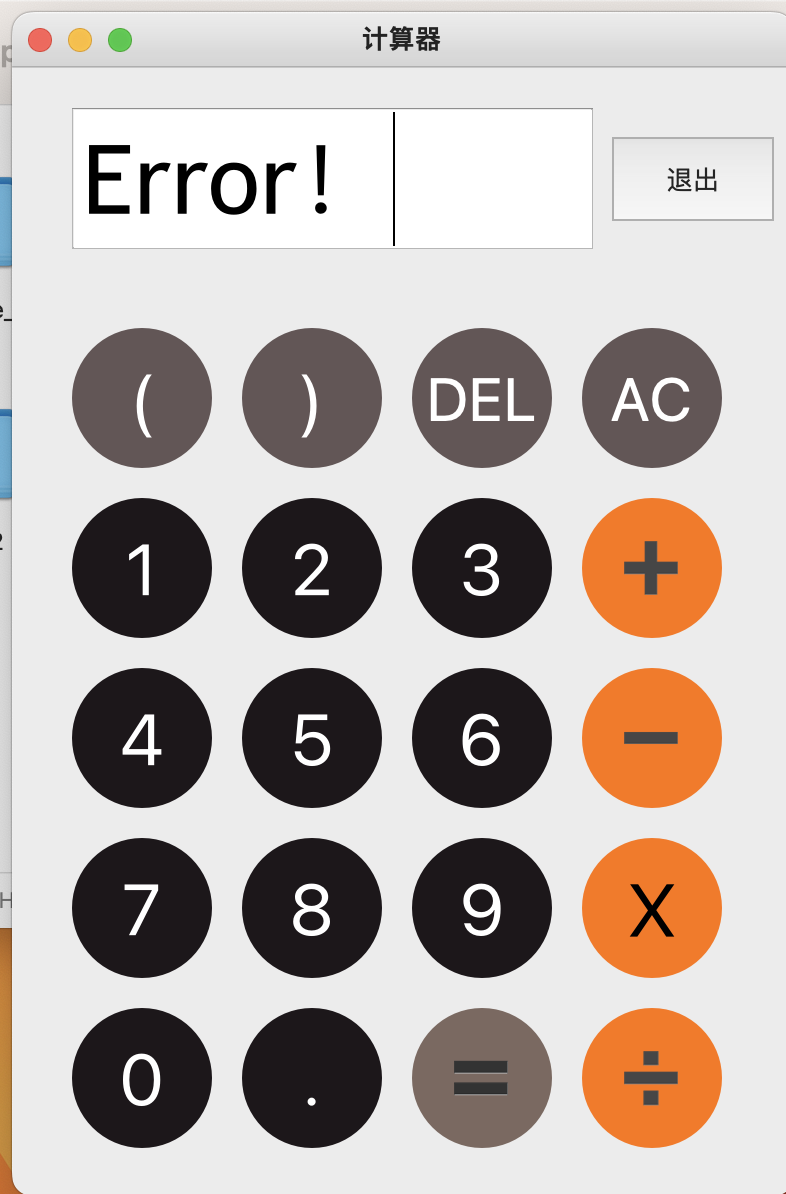
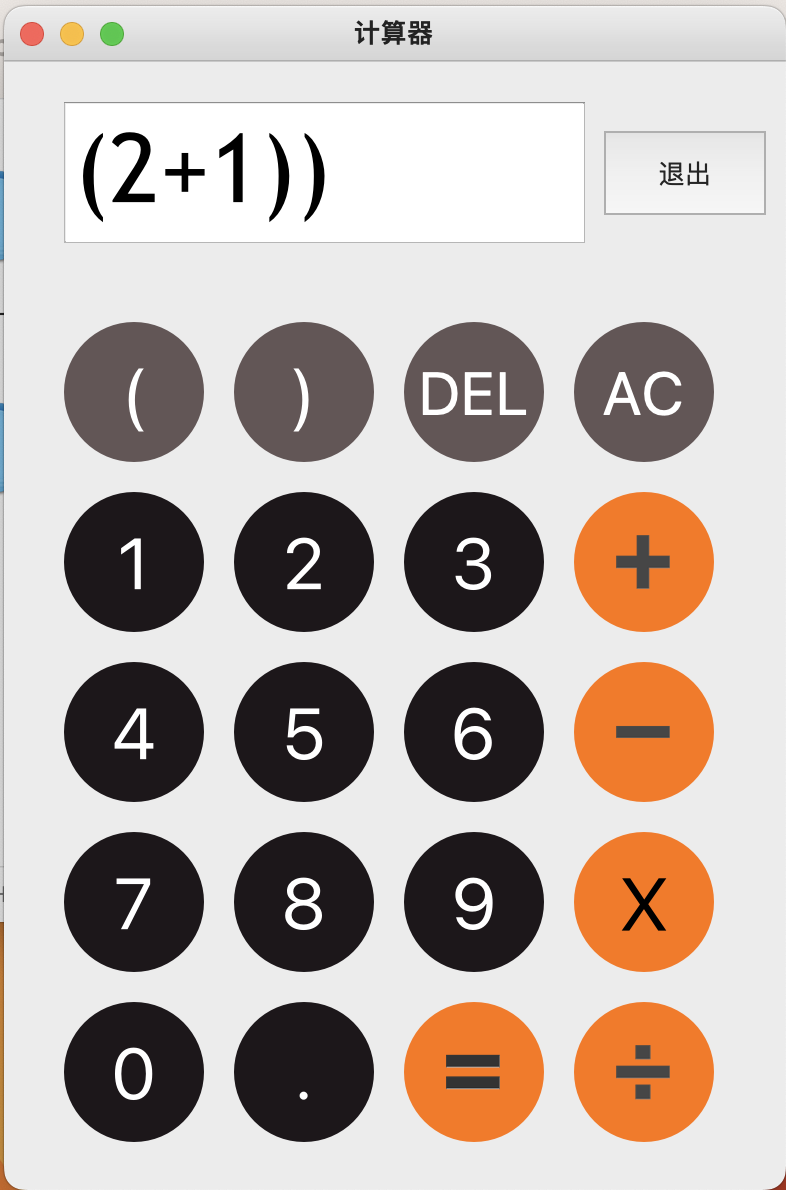
点击del后：14变为1



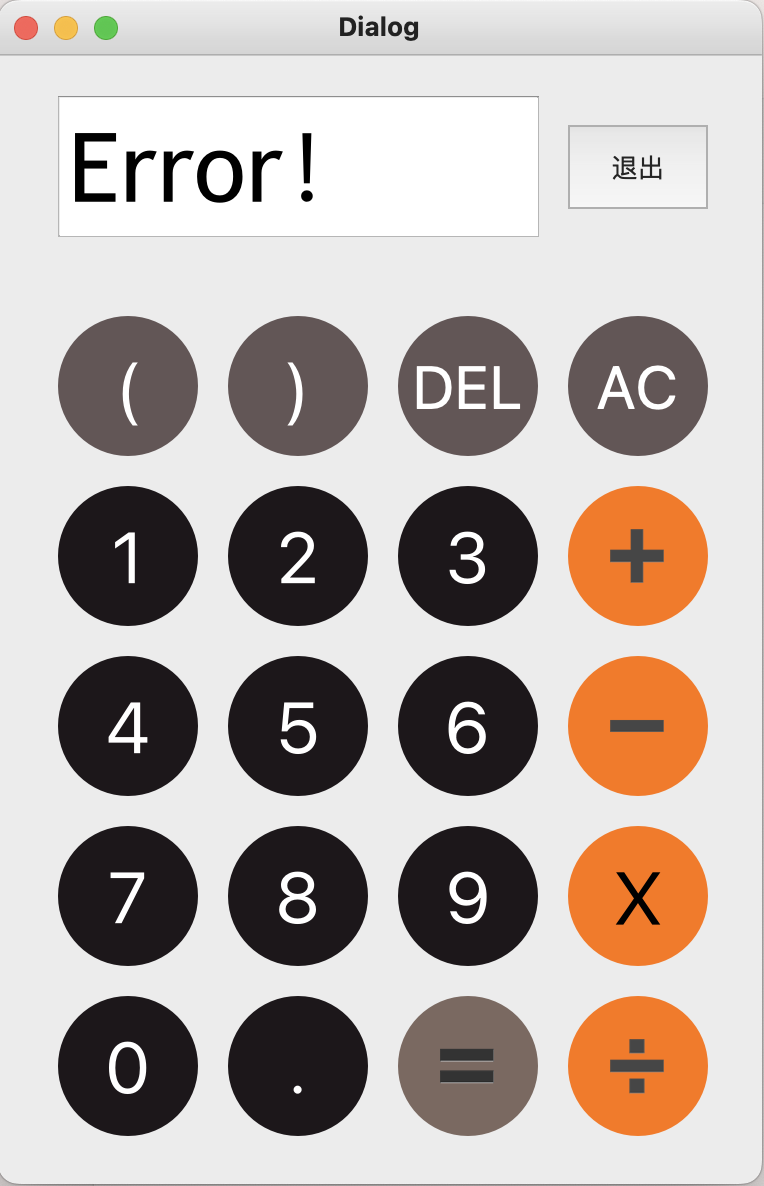
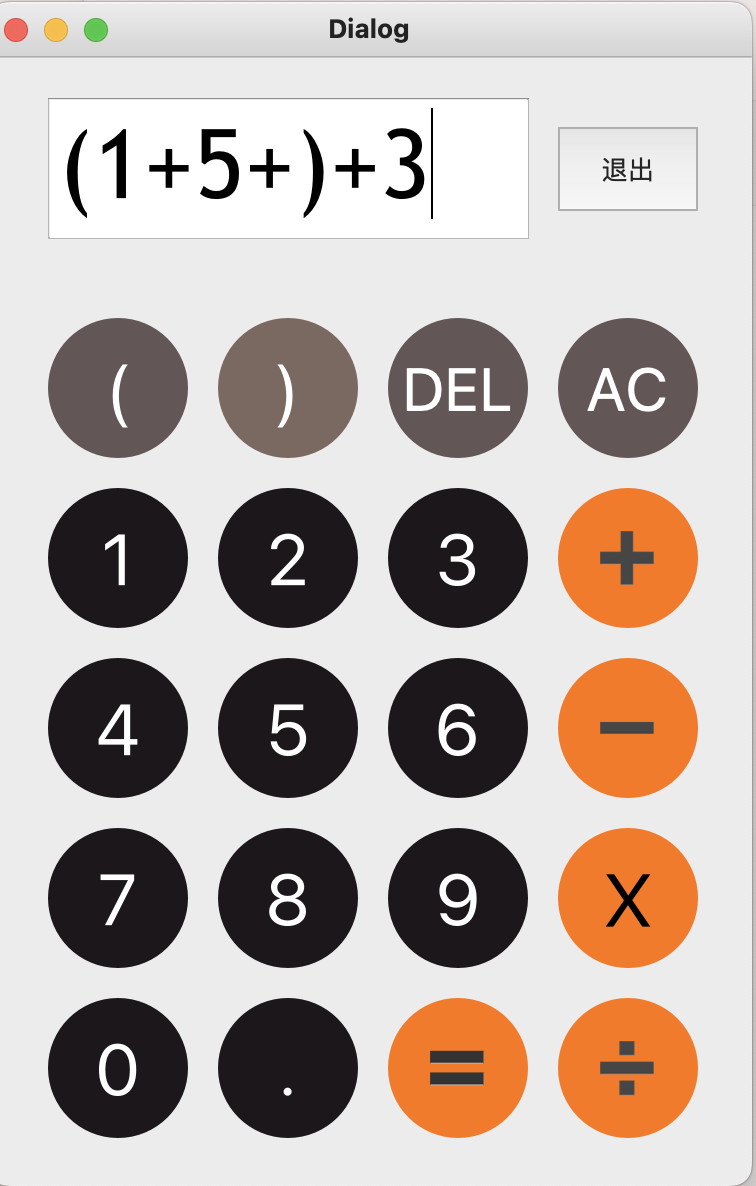
点击AC后，1354变为0

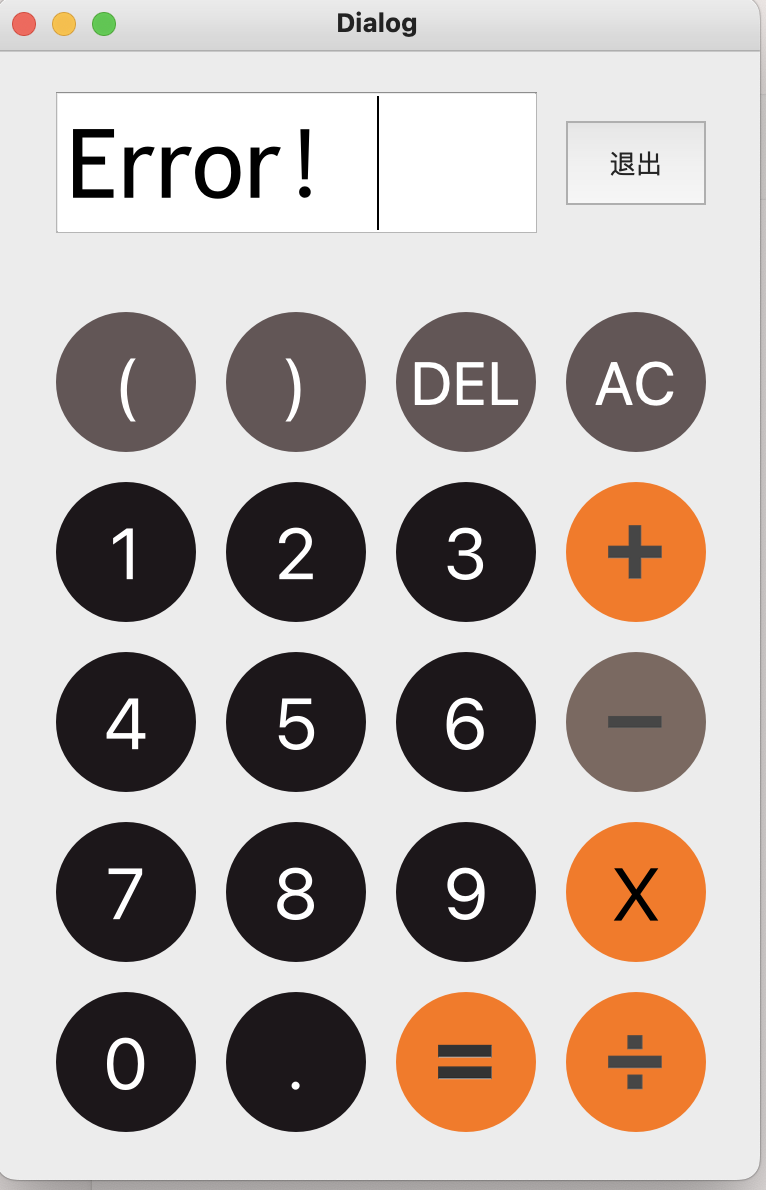
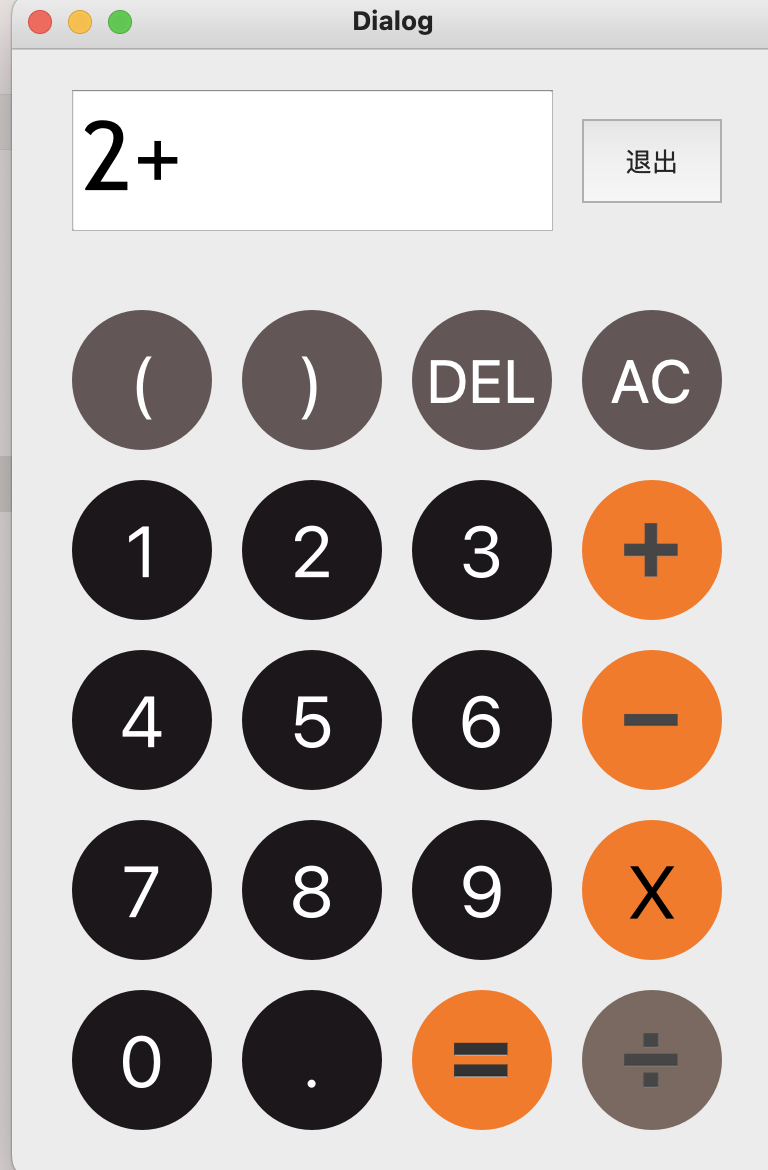


括号不匹配时：

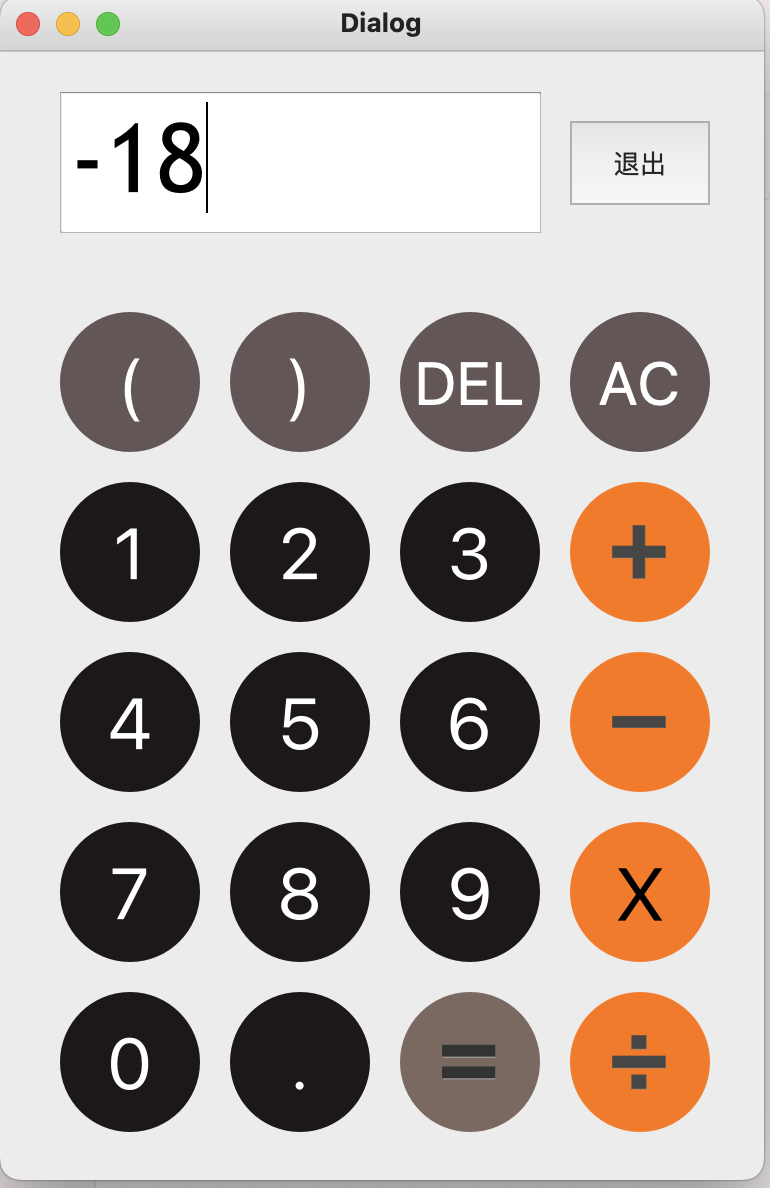
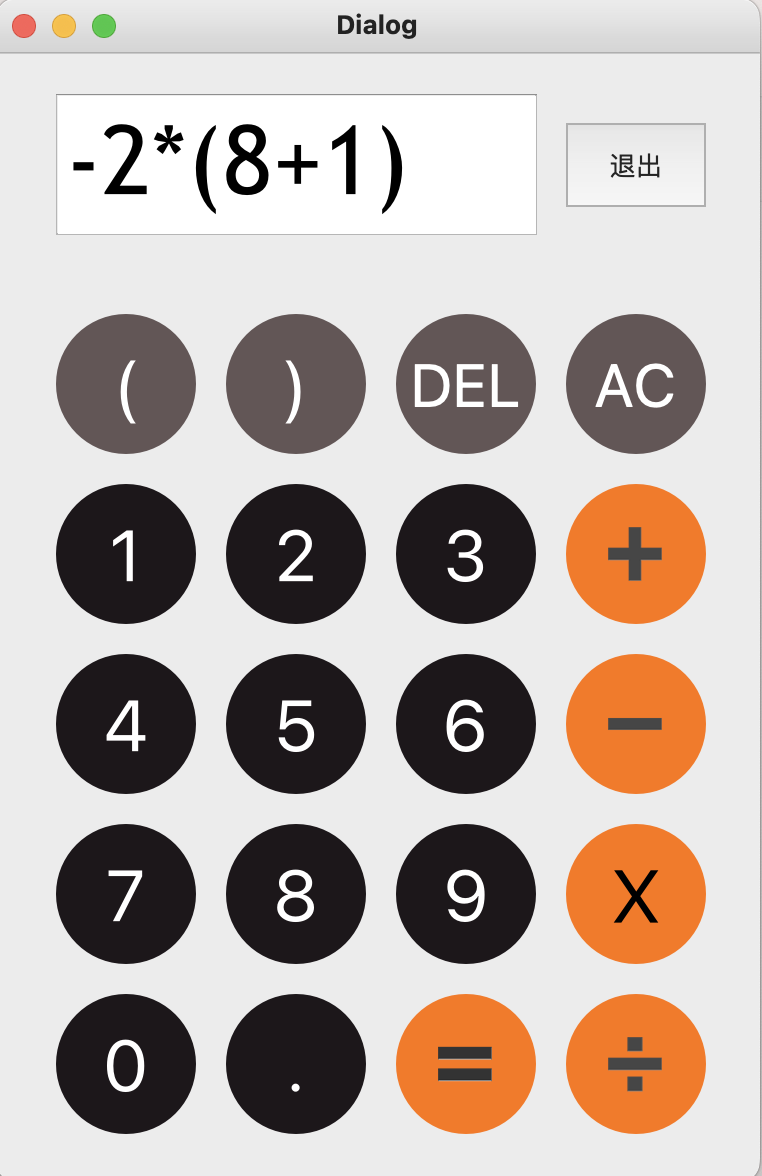


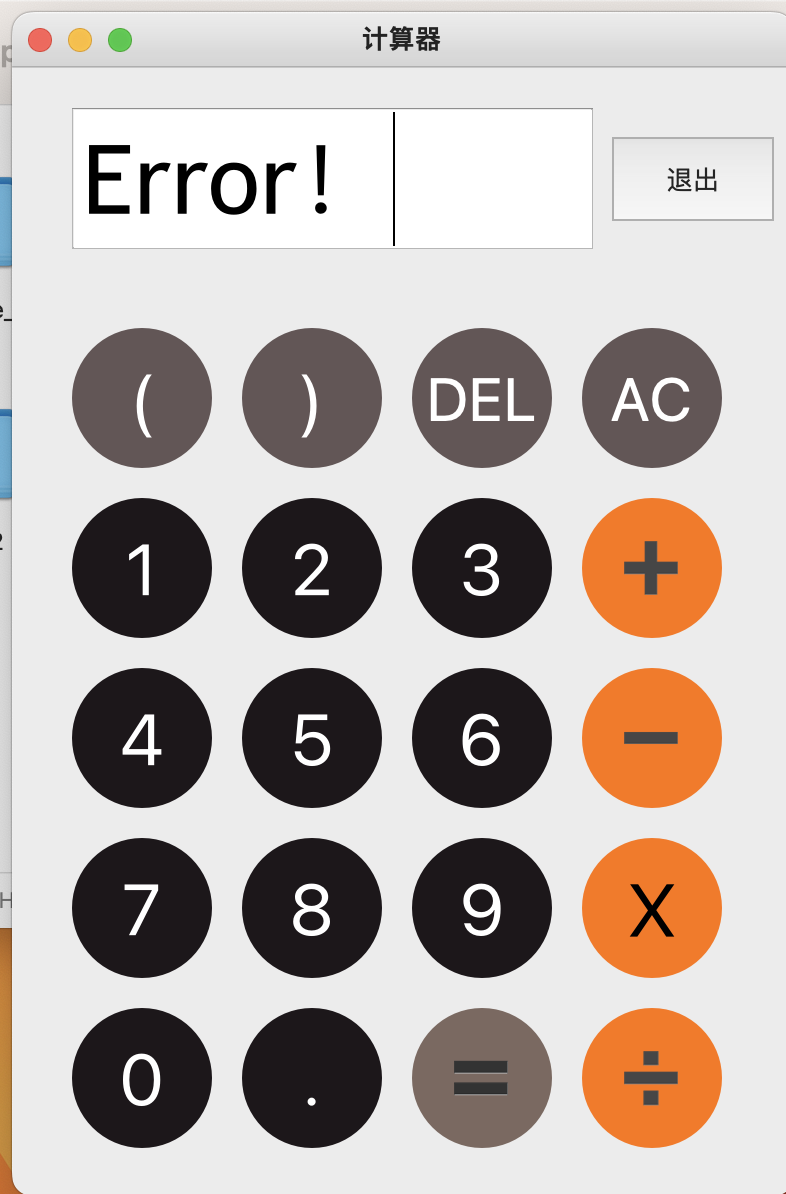
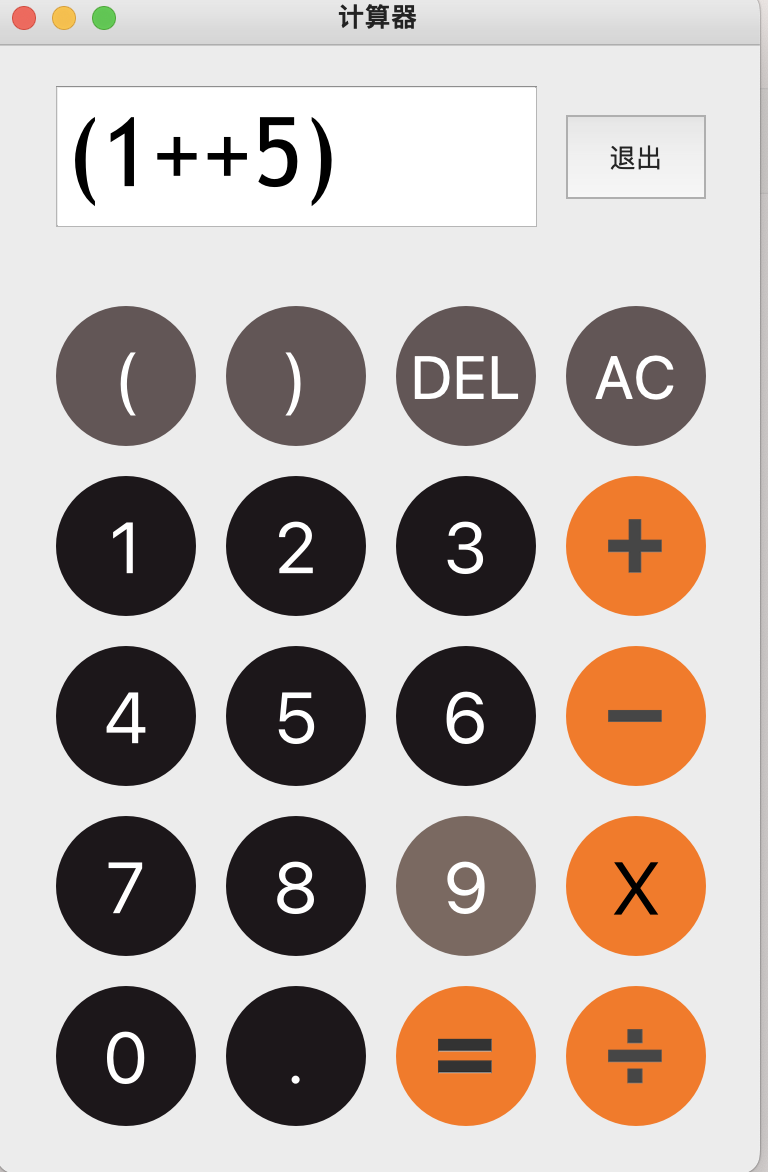
输入格式错误时



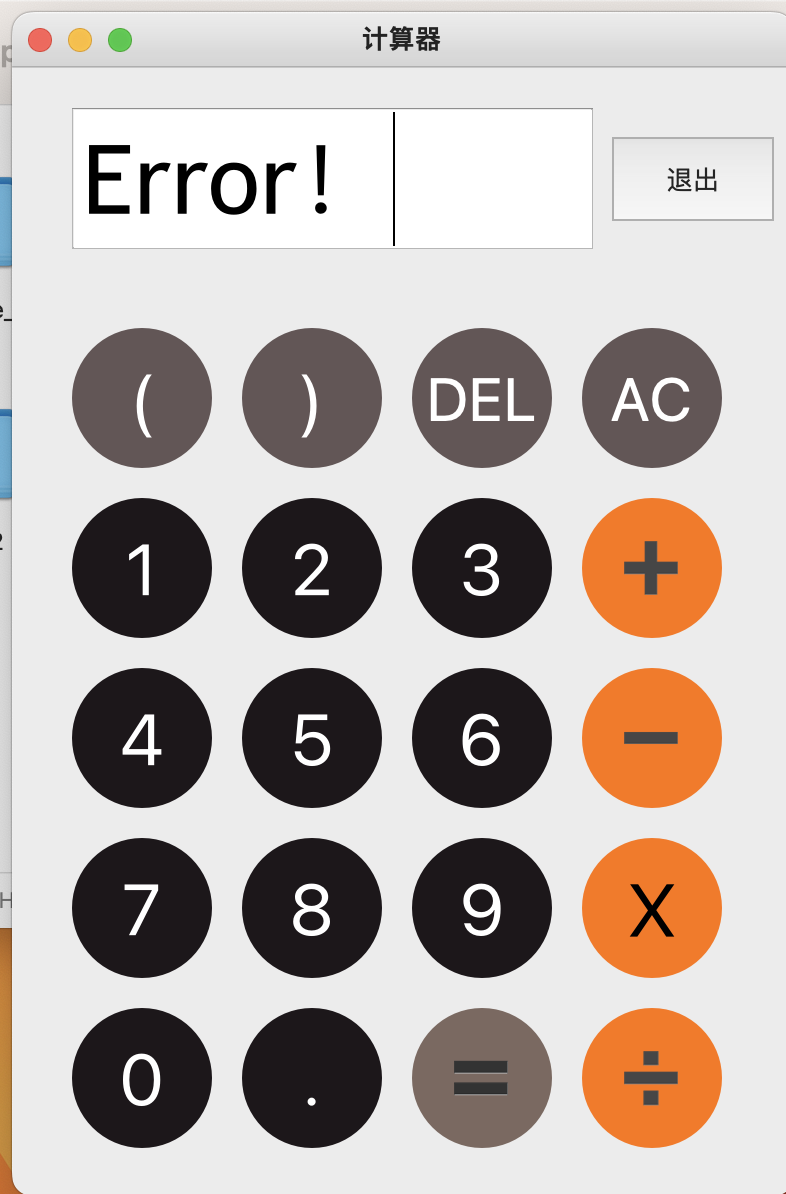
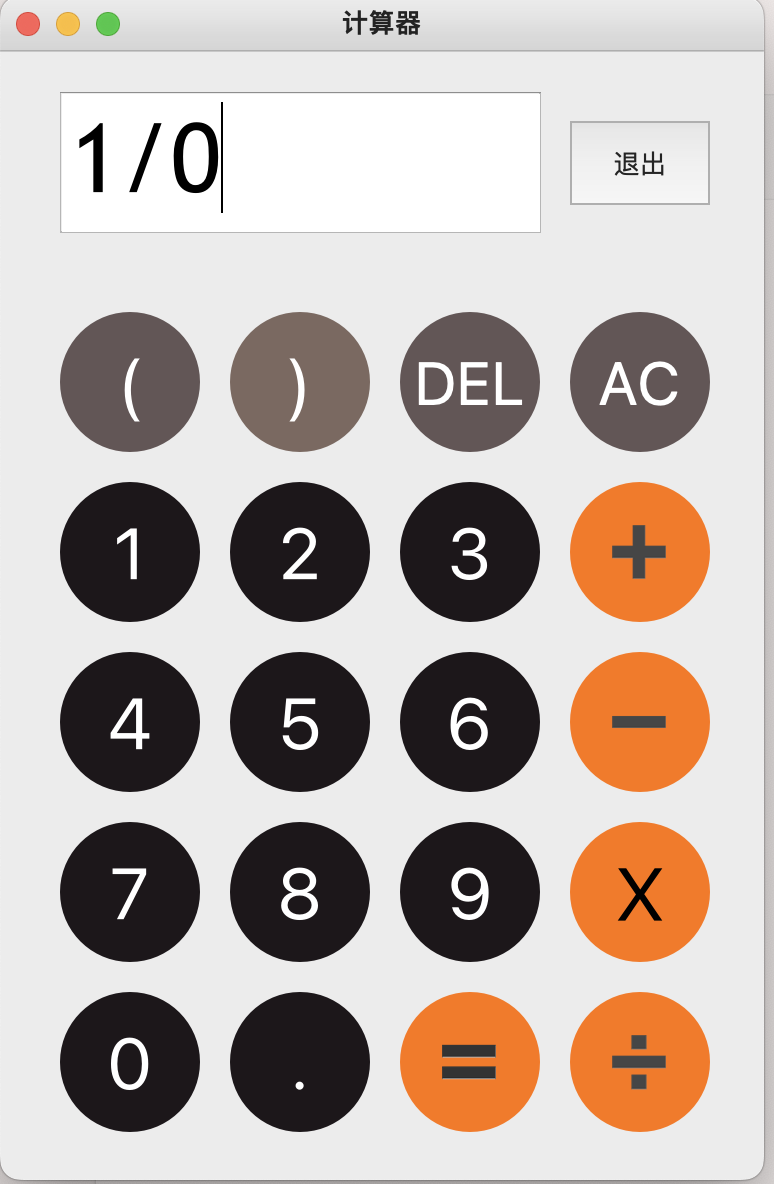


负数：





除数为0



**八、总结**

本次实验，我使用栈来实现计算器。在实验中，这个计算器将中缀表达式转换为后缀表达式，然后使用栈来计算后缀表达式的值。

这个计算器使用栈来实现中缀表达式的转换和后缀表达式的计算，结合GUI界面提供用户输入和输出结果的交互。作为一个初学者，在完成这个用栈实现计算器的实验过程中，我对Python、QTdesigner和数据结构栈有了更深入的理解。

我了解了Python中的模块的概念，知道如何调用已经写好的类和函数，加快了代码编写速度，同时也使得代码更加易读易懂。同时，QT designer是一个非常好用的GUI设计工具，可以轻松地进行UI设计和布局。通过QT designer，我了解了信号与槽的概念和用法。同时，也学会了如何将设计好的UI界面和Python代码进行连接。

栈，是实验中最重要的部分，也是我最为陌生的一部分。通过这次实验，我了解了栈的基本概念和特点，学会了栈的实现方式以及如何使用栈进行表达式的计算，也对栈的应用有了更深刻的理解。

通过这次实验，我不仅学到了Python编程、QT designer界面设计、数据结构——栈等知识，还锻炼了自己的逻辑思维和编程能力，收获颇丰。我会在以后的学习和实践中继续深入学习和应用这些知识。