计算器制作

实验目的

使用C++制作一个计算器，拥有基本的计算能力，同时能够对错误的用户输入做出响应与提示。

实验方法

本项目一开始是使用vs2022+控制台窗口的形式实现，后改进为基于Qt框架进行开发，类型是Qt Widgets Application，通过Qt creator手动制作简单的计算器界面，并通过信号与槽实现逻辑处理，涉及的编程语言为C++。项目已打包并上传至github平台，clone链接为：
https://github.com/liangjunmeng/calculator_Qt.git

实验环境

Qt Creator 17.0.0，Qt 6.5.3，C++17，visual studio 2022

实验内容

先使用vector容器读取用户输入的字符串，由于用户输入的是中缀表达式，故在读取阶段需要将里面的运算数和符号分离，分别存储，运算数包括正负整数、小数和用科学记数法表示的数（如2e1、2e+2、2E-3）；然后通过符号栈和运算符优先级的比较将该中缀表达式转换为后缀表达式，同样将结果存储在vector容器中；最后使用栈来计算后缀表达式，并将结果输出，每个阶段都有对用户输入的检查，以验证用户输入是否正确。

计算器最后能实现的功能包括加减乘除、取余、次方求值和阶乘、删除、清空。

该项目的实现主要分为四部分：中缀表达式的读取、中缀转后缀、计算后缀表达式、输入检错。

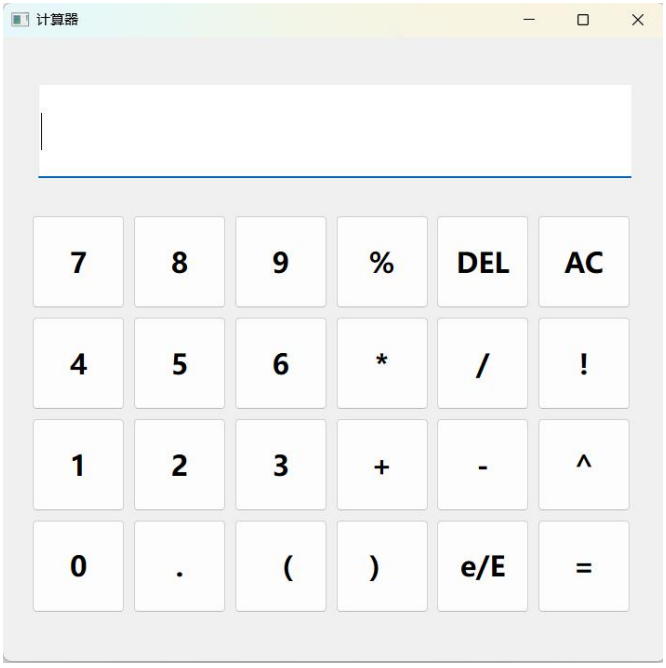
①中缀表达式的读取主要需要分清运算数的起始位置和终止位置，尤其是对小数和科学计数法数字的读取，可以通过for循环嵌套和if判断语句实现；

②中缀转后缀的原理为：读取中缀表达式时，如果是运算数，则直接放入后缀表达式中；如果是运算符，则：如果栈为空或者运算符优先级高于栈顶元素，直接入栈，否则弹出栈顶元素直至满足入栈条件；如果是左括号，直接入栈，且其优先级最低；如果是右括号，则不断弹出栈顶元素至后缀表达式中，直到遇到左括号，弹出该左括号，当然，左右括号都不会放入到后缀表达式中。

③计算后缀表达式则较为简单：不断检索后缀表达式（从左至右，即 vector 容器从下标 0 到末尾），如果遇到一元运算符，则将前一个元素取出用于计算，如果遇到二元运算符，则取出前两个元素用于计算，运算符和运算数都需要删去，最后放入 vector 容器中，如果用户输入式子正确，最后 vector 容器只会有一个元素，即最终运算结果。

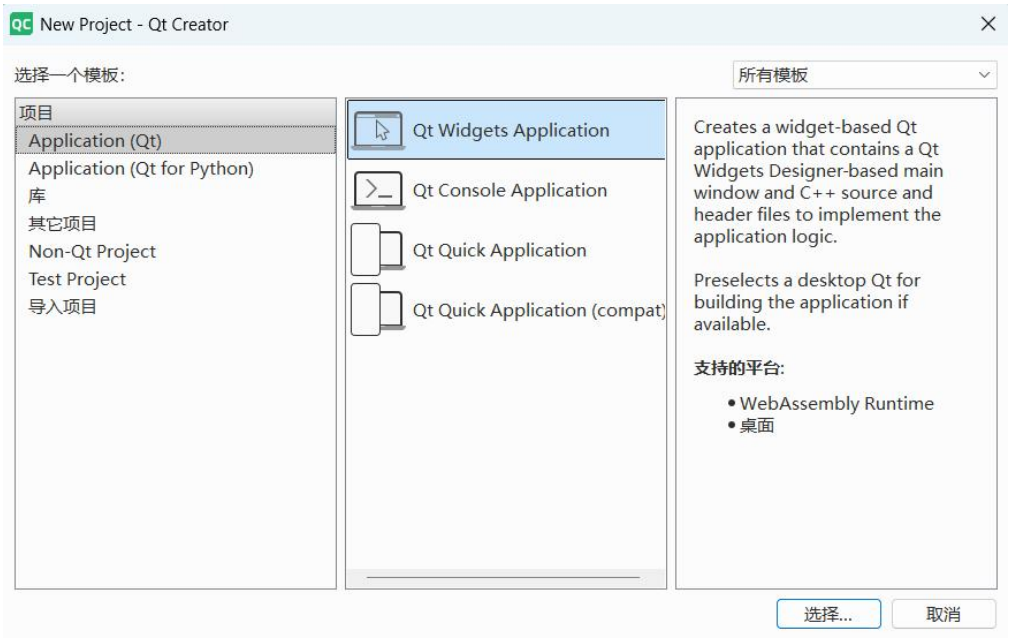
④输入检错在后面的实验步骤中作说明。

计算器 ui 界面如下：

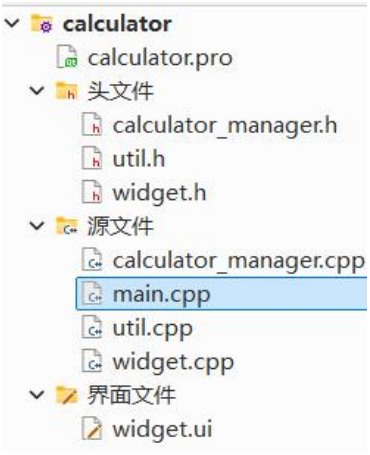


实验步骤

1、首先打开 Qt creator 并创建 Qt Widgets Application 项目，取名为 calculator



进入项目后系统会自动生成 calculator.pro、widget.h、widget.cpp、main.cpp、widget.ui 文件，其中 main.cpp 是主函数，通过声明窗口部件类的派生类 widget 并执行其 show 方法展示窗口。这里先创建好需要的头文件和源文件：



其中 calculator_manager 为最核心的类，不仅用于读取输入，还能实现中缀表达式转后缀等功能，util 则为工具类，一些常用的函数，如判断字符串是否是合法运算数、返回运算符优先级等在其内部定义，声明为 static，便于直接使用。

头文件内容如下：

calculator_manager.h

```
class CalculatorManager {
public:
    CalculatorManager();
    QString getInput(std::string str); // 录入字符串
    bool toInfixExpr(); // 将用户输入转换为中缀表达式进行存储
    bool toPostfixExpr(); // 将中缀表达式转换为后缀表达式
    void printInfixExpression(); // 打印中缀表达式
    void printPostfixExpression(); // 打印后缀表达式
    void clearAll(); // 清空数据成员
private:
    std::string input; // 用户输入字符串（符号形式）
    std::vector<std::string> infixExpression; // 用户输入字符串（中缀表达式形式）
    std::vector<std::string> postfixExpression; // 后缀表达式
    double result; // 计算结果
};
```

util.h

```
// 工具类
class Util {
public:
    Util() = delete; // 禁止实例化
    static bool isRight(const std::string& str, int pos); //判断用户输入是否合法
    static int getPriority(const char &code); // 获得符号优先级
    static bool isNumber(const std::string& str); // 判断字符串是否为合法数字
    // 判断在中缀表达式转后缀时，操作符是否能入符号栈
    static bool canGetInStack(const std::stack<std::string>& marks, std::string str);
    // 计算后缀表达式，并将结果存至result中
    static bool countPfExpr(std::vector<std::string> postfixExpression, double& result);
};
```

下面进行对成员函数的详细说明

2、Util

①isRight 函数用于判断用户输入字符的正确性，还会检查数字是否于右括号紧挨着，若紧挨着则输入非法，为什么在这里检查呢？因为形如(2)2 的式子在中缀转后缀或后缀计算中都显示正常并能有结果

```
bool Util::isRight(const std::string& str, int pos) {
    if ((str[pos] <= '9' && str[pos] >= '0') || str[pos] == '.' || str[pos] == 'e' || str[pos] == 'E') {
        // 避免(2)2被判断为正确值
        if (pos != 0) {
            return str[pos - 1] != ')';
        }
        else {
            return true;
        }
    }
    else if (str[pos] == '+' || str[pos] == '-') {
        return true;
    }
    else if (str[pos] == '*' || str[pos] == '/' || str[pos] == '%') {
        return true;
    }
    else if (str[pos] == '(' || str[pos] == ')') {
        return true;
    }
    else if (str[pos] == '^' || str[pos] == '!') {
        return true;
    }
    else {
        return false;
    }
}
```

②getPriority 函数用于返回运算符或左括号的优先级：

```
int Util::getPriority(const char& code){
    switch (code)
    {
    case '(': return 0;
    case '+':
    case '-': return 1;
    case '*':
    case '/':
    case '%': return 2;
    case '^': return 3;
    case '!': return 4;
    default: return -1;
    }
}
```

③isNumber 函数用于判断字符串是否代表着运算数，可通过 C++的正则匹配库实现：

```
bool Util::isNumber(const std::string& str){
    // 匹配规则
    std::regex pattern(R"(^[+-]?\d+(\.\d+)?([eE][+-]?\d+)?$)");
    return std::regex_match(str, pattern);
}
```

④canGetInStack 函数用于中缀表达式转后缀时判断运算符是否能入栈，if 中的逻辑在上面实验内容部分已经说明过（比较运算符优先级等）：

```
bool Util::canGetInStack(const std::stack<std::string>& marks, std::string str) {
    // 如果栈为空、操作符为左括号或操作符优先级高于栈顶符号，允许入栈
    if (marks.empty() || str == "(" || Util::getPriority(str[0]) > Util::getPriority(marks.top()[0])) {
        return true;
    }
    else {
        return false;
    }
}
```

⑤countPfExpr 函数用于计算后缀表达式，原理也已在上面的实验内容部分说明，主要是先判断字符串是否是运算数，然后使用栈来不断的遍历计算，使用了一些方便计算的自带库函数，如 fmod，可以对 double 类型的数字进行互相取余操作：

```
bool Util::countPfExpr(std::vector<std::string> postfixExpression, double& result) {
    std::stack<double> countNum; // 便于计算的栈
    double num1, num2;
    for (int i = 0; i < (int)postfixExpression.size(); i++) {
        if (Util::isNumber(postfixExpression[i])) {
            // 将字符串转换为double类型的数字
            countNum.push(std::stod(postfixExpression[i]));
        }
        else {
            if (countNum.size() < 2 && postfixExpression[i] != "!") {
                return false;
            }
            num1 = countNum.top();
            countNum.pop();
            // 对于一元运算符"!"来说，只需要弹出栈的一个元素用于计算
            if(postfixExpression[i] != "!"){
                num2 = countNum.top();
                countNum.pop();
            }
            if (postfixExpression[i] == "+") {
                num2 += num1;
            }
            else if(postfixExpression[i] == "-") {
                num2 -= num1;
            }
            else if (postfixExpression[i] == "*") {
                num2 *= num1;
            }
            else if (postfixExpression[i] == "/") {
                if (num1 == 0) {
                    return false;
                }
                // 栈顶元素是除数
                num2 /= num1;
            }
            else if(postfixExpression[i] == "%"){
                if (num1 == 0) {
                    return false;
                }
                num2 = fmod(num2, num1);
            }
            else if(postfixExpression[i] == "^"){
                num2 = std::pow(num2,num1);
            }
            else if(postfixExpression[i] == "!"){
                // 判断被阶乘的是否是整数，否则报错
                if(!(std::fmod(num1, 1) == 0.0)){
                    return false;
                }
                int num22 = 1;
                for(int i = 1; i <= num1; i++){
                    num22 *= i;
                }
                num2 = num22;
            }
            countNum.push(num2);
        }
    }
    if (countNum.size() != 1) {
        return false;
    }
    result = countNum.top();
    return true;
}
```

3、CalculatorManager

①构造函数：初始化各数据成员

```
CalculatorManager::CalculatorManager() {
    input = "";
    infixExpression = {};
    postfixExpression = {};
    result = 0;
}
```

②getInput 函数，用于对用户输入的处理，由于是对传入的参数进行处理，故不算是读取输入操作，在处理前需要清空所有数据成员，然后判断是否为空、转为中缀表达式进行存储、转为后缀表达式进行存储、计算后缀表达式，最后将 double 类型的结果转换为 QString 类型，便于在 Qt 界面中显示，由于以上步骤返回结果为 bool 类型，故可以在每个阶段对用户输入进行检查：

```
QString CalculatorManager::getInput(std::string str) {
    clearAll();
    input = str;
    if (input.empty()) {
        return "用户未输入！";
    }
    if (!toInfixExpr()) {
        return "用户输入非法！";
    }
    if (!toPostfixExpr()) {
        return "用户输入非法！";
    }
    if (!Util::countPfExpr(postfixExpression, result)) {
        return "用户输入非法！";
    }
    else {
        return QString::number(result);
    }
}
```

③toInfixExpr 函数，将用户输入的字符串进行分割，筛选出运算数和运算符首先声明几个变量，record 用于存储分割过程中的运算数，pointNum 用于计算小数点的数量，eNum 用于计算 e 或 E 的数量，方便判断用户输入的合法性，如 2..和 2ee 就肯定是错的，然后 for 循环对输入字符串进行逐一检查：

```
std::string record;
int i, j, pointNum, eNum;
for (i = 0; i < (int)input.size(); i++) {
```

使用工具类的检错函数进行初步检错：

```
// 非法输入
if (!Util::isRight(input, i)) {
    return false;
}
```

对正负号的特殊情况进行处理，如当正负号位于开头位置或前面是左括号时，将其等价于前面有个 0，如-1*2 等价于 0-1*2、(-1*2)等价于(0-1*2)：

```
// 避免-1、+1、+(1+1)、1+(+(1+1))被判断为非法字符
if ((input[i] == '+' || input[i] == '-')) {
    if (i == 0 || input[i - 1] == '(') {
        infixExpression.push_back("0");
    }
}
```

随后判断字符是否为数字，若不为数字则直接存入中缀表达式（vector）中，若为数字则开始检查该运算数的终止位置（整个项目中比较难的地方）：

```
record = "";
// 记录数字，包括整数和小数（不包括负数），而是将符号和数字两部分分别存储
if (std::isdigit(static_cast<unsigned char>(input[i]))) {
```

利用内嵌 for 循环进行遍历检查：

```
pointNum = 0; // 小数点数量
eNum = 0; // e的数量（科学计数法）
record.push_back(input[i]);
// 判断数字截至位置，同时查看是否是小数
for (j = i + 1; j < (int)input.size(); j++) {
```

如果遍历到的字符是数字，则添加到 record 中，方便最后将整个运算数存入中缀表达式，否则判断是否为小数点、e、E；若不是以上符号，则数字终止位置即为下标 j，若为小数点，先判断小数点是否已经出现或是否是整个字符串的最后一个字符，是则说明用户输入有误，否则加入至 record 中，并将 pointNum 加一，对 e 或 E 的处理类似于小数点，只不过这里直接判断 e 或 E 的下一个字符是否合法（数字、正负号为合法，其他都不合法），最后刷新初识下标 i：

```
if (!std::isdigit(static_cast<unsigned char>(input[j]))) {
    if (input[j] != '.' && input[j]!='e' && input[j]!='E') {
        i = j - 1; // for循环一轮结束后会自动对i进行自增
        break;
    }
    else if(input[j] == '.'){
        // 小数点过多或检索到最后一个字符才是小数点时，说明用户输入有问题，返回false
        if (pointNum == 1 || j == (int)input.size() - 1) {
            return false;
        }
        else {
            record.push_back('.');
            pointNum++;
        }
    }
    else if(input[j] == 'e' || input[j] == 'E'){
        // e过多或检索到最后一个字符才是e时，说明用户输入有问题，返回false
        if (eNum == 1 || j == (int)input.size() - 1 ) {
            return false;
        }
        else {
            // 直接检索e后面的字符，不为数字或正负号则直接判为非法输入
            if(input[j+1]!='+' && input[j+1]!='-' && (input[j+1] < '0' || input[j+1] > '9')){
                return false;
            }
            else{
                record.push_back(input[j]);
                record.push_back(input[j+1]);
                eNum++;
                j++;
            }
        }
    }
}
```

④toPostfixExpr 函数，实现对中缀表达式到后缀表达式的转换，这里的逻辑非常简单，参考实验内容部分的原理即可，需要判断左右括号的匹配是否正常，若不匹配，则在弹出符号栈的元素时，会弹到栈为空也无法找到左括号：

```
bool CalculatorManager::toPostfixExpr() {
    if (infixExpression.size() == 0) {
        return false;
    }
    std::stack<std::string> marks; // 符号栈
    for (int i = 0; i < (int)infixExpression.size(); i++) {
        // 如果是运算数，直接放入后缀表达式中
        if (Util::isNumber(infixExpression[i])) {
            postfixExpression.push_back(infixExpression[i]);
        }
        else if (infixExpression[i] == ")") {
            while (true) {
                // 缺少左括号
                if (marks.empty()) {
                    return false;
                }
                if (marks.top() == "(") {
                    marks.pop();
                    break;
                }
                else {
                    postfixExpression.push_back(marks.top());
                    marks.pop();
                }
            }
        }
        else {
            // 如果栈为空或待处理操作符优先级高于栈顶符号，直接入栈
            if (Util::canGetInStack(marks,infixExpression[i])) {
                marks.push(infixExpression[i]);
            }
            else {
                while (!Util::canGetInStack(marks, infixExpression[i])) {
                    postfixExpression.push_back(marks.top());
                    marks.pop();
                }
                marks.push(infixExpression[i]);
            }
        }
    }
    while (!marks.empty()) {
        postfixExpression.push_back(marks.top());
        marks.pop();
    }
    return true;
}
```

⑤剩下的几个函数，如用于测试的打印中后缀表达式和清空函数，就是简单的遍历打印或赋值为空串等简单操作。

3、widget 类中添加几个数据成员，便于对字符串的处理：

```
QString inputText;
CalculatorManager calculatorManager;
```

实现 ui 界面的逻辑功能，先设计好 ui，如按钮是 Buttons 里的 Push Button 类型、显示器是 Input Widgets 里的 Line Edit 类型，使用栅格布局将界面变得整齐且美观，对于每个按钮，只需要鼠标右键点击然后点击“转到槽”即可实现对每个按钮的处理（响应）函数的声明：

对于按钮 0、+、(、e 等符号，处理逻辑都一样，如对 0 的处理，首先需要判断显示器里的信息是不是“用户输入非法！”或“用户未输入！”，不是则添加符号 0 到显示器中，然后重置显示器内容为上一次的正确运算结果，避免符号 0 被加到“用户未输入！”这类字符串中，同时还会查找光标位置，将字符插入到光标处，最后获得显示器焦点，让光标得以一直显示：

```
void Widget::on_button0_clicked()
{
    // 如果用户输入有误则不对用户之后的第一次输入做出回应，避免对报错提示进行编辑
    if(!ui->screen->text().startsWith("用")){
        // 光标位置
        int pos = ui->screen->cursorPosition();
        inputText.insert(pos, "0");
    }
    ui->screen->setText(inputText);
    ui->screen->setFocus();  // 确保QLineEdit获得焦点
}
```

对于 DEL 按钮，和上面处理大同小异，只不过是做删除操作：

```
void Widget::on_buttonDel_clicked()
{
    int pos = ui->screen->cursorPosition();
    if(inputText.size()>0 && !ui->screen->text().startsWith("用")){
        if(pos>0){
            inputText.remove(pos-1, 1);
        }
    }
    ui->screen->setText(inputText);
    ui->screen->setFocus();
}
```

对于 AC（清除）按钮，直接清空所有数据即可：

```
void Widget::on_buttonAC_clicked()
{
    inputText.clear();
    ui->screen->clear();
    ui->screen->setFocus();
}
```

如果按下等于号，会先判断显示器内容，如果是计算式，则调用CalculatorManager对象进行处理并返回结果，否则代表出错，显示上一次正确计算结果：

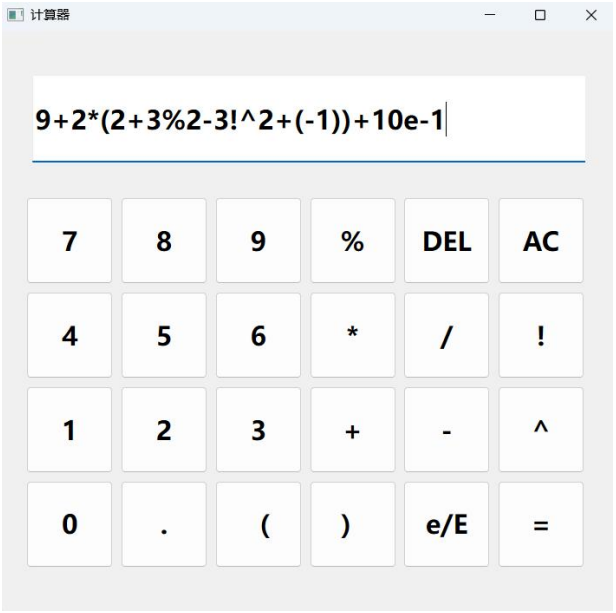
```
void Widget::on_buttonEqual_clicked()
{
    QString str = "";
    str = calculatorManager.getInput(ui->screen->text().toStdString());
    // 如果用户输入有问题，保留上一个无问题的计算结果
    if(!str.startsWith("用")){
        inputText = str;
    }
    ui->screen->setText(str);
    ui->screen->setFocus();
}
```

在Widget构造函数里面连接信号与槽，实现按下回车等同于按下等于号，但是显示器上有光标按下回车才会有效果，所有按下其他按钮要始终让光标显示：

```
//连接信号与槽，实现回车等效于按下等于号
connect(ui->screen, SIGNAL(returnPressed()), this, SLOT(on_buttonEqual_clicked()));
```

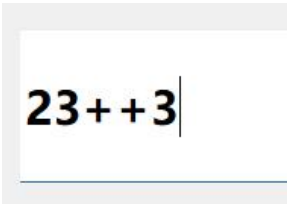
实验结果

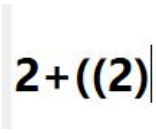
按下ctrl+r即可直接运行

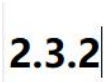


显示：

运算正确







显示：

同时对于较大数字的运算，比如：

结果显示为 0，因为数字已经超过计算机中 double 存储的上限

实验感想

通过此次项目实验，加深了我对 c++基础语法的理解以及学会了 Qt 的简单应用，但是实现的计算器还有不足之处，比如三角函数等运算没有实现。实验过程中，也有对 bug 的修复，比如右括号后面跟数字却显示正确、结果显示 e（科学计数法）后计算器无法计算等、一开始只有简单的加减乘除、阶乘是一元运算符导致栈越界等，后面通过改正和添加新内容修复了 bug 并丰富了功能。