面向对象程序设计

课程报告

班级：231223

学号：20221000983

姓名：陈子逸

目录

一　课程设计题目与要求

选择题目一：仿照 Windows 系统的计算器软件，设计通用计算器界面，开发一款实用的计算器软件。

Windows 10系统计算器界面及功能如下：

可见，除了解决一般加减乘除问题外，还需有更多功能的实现，如清零、历史记录、平方等。在之前的课程作业中，也有实现三角函数的要求等等，如下图所示：

链栈是一种采用链式结点（node）结构实现栈（stack）后进先出（LIFO），只能在顶部结点进行操作的数据结构。基于链栈结构实现一个简单计算器的功能。

【要求】

1. 支持加、减、乘、除、求余、括号、次幂等基本操作。

2. 扩展计算器功能，使其支持 sin、cos、tan、sqrt 等功能。

3. 基于设计的计算器，计算以下表达式，其中 pi 的值可以通过 std::atan(1.0)*4 获得:

(1) 3-2*4+(6-1)/2+5

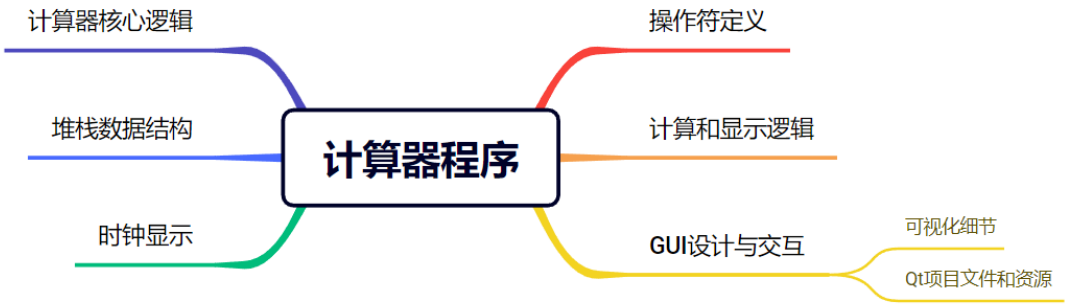
(2) sin(pi/6)+ cos(sqrt(2)) ^ tan(pi/3)

二　需求分析

2.1 问题描述

用 c++程序设计计算器，需要通过栈等数据结构解决数据存储问题，再通过各种算法解决简单运算、各种函数、功能实现。主要问题在于如何组织各类数据，灵活调用各种算法，同时解决多种算法在同一运算语句中的优先级，以及错误计算的应对。

同时，在 qt 中需要实现对话框之间的信息传递，也就是要通过信号与槽的机制来传递信息或者实现控制功能。



2.2 系统环境

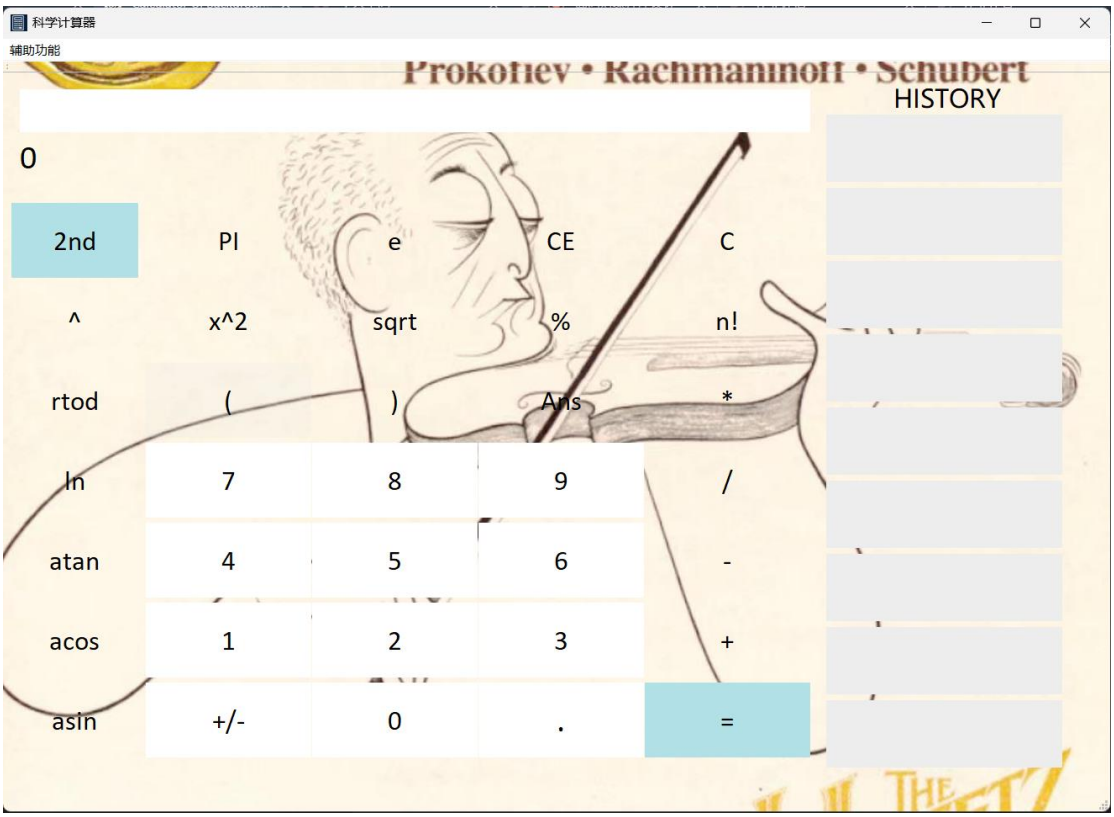
Microsoft Visual C++ (MinGW.org GCC-6.3.0-1) 6.3.0
Qt 5.14.2 MinGW-64bit

2.3 运行要求

Windows 系统下运行，Win XP,Win7\Win8\Win10\Win11 等 Windows 系统版本下运行

三　具体设计

效果预览：



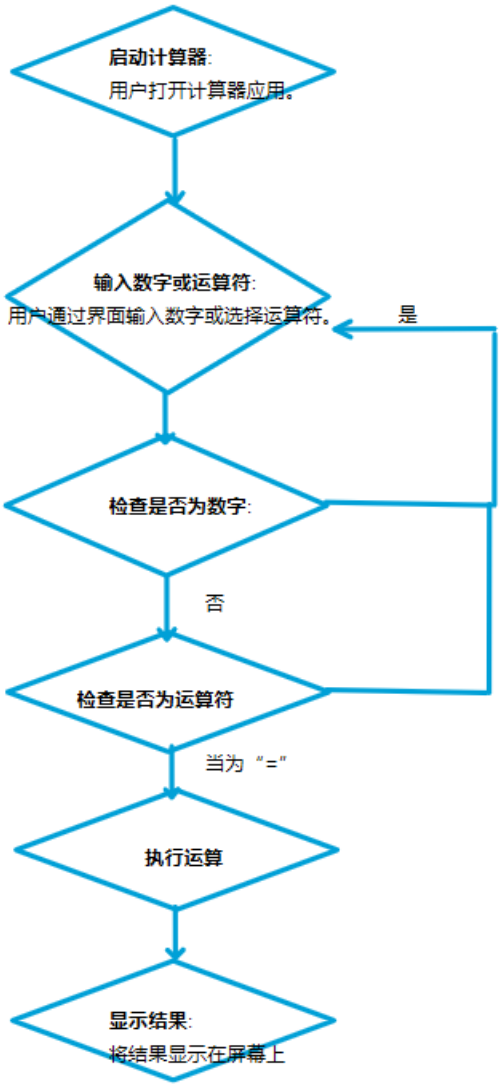
（一）工作流程

程序启动后，主界面展现，其中顶部是一个用于显示数学表达式的标签（QLabel）。标签下方装配有一个专门显示当前时间的数字时钟（QLCDNumber），用于实时展示时间。界面的右侧部分设有一个用于展示历史计算记录的文本区域（QTextBrowser），用户可以在此查看之前的计算历史。

按键排列在表达式显示区的下方，用户可以通过这些按键输入表达式。界面支持在标准和科学模式之间切换，科学模式提供更多的数学运算符和功能，如三角函数、对数等。

用户交互流程：

1. **表达式输入**：用户在界面中输入数学表达式的过程是互动的，可以通过点击界面上的数字和运算符按钮来构建表达式，或者直接使用键盘输入，增加用户操作的灵活性。每次按钮的点击都会即时反馈到显示区域，使用户可以即时看到自己的输入。
2. **计算执行**：用户完成表达式输入后，点击等号（=）按钮发起计算请求。此时，程序会调用解析函数，解析整个表达式，识别其中的数字、运算符及其优先级。
3. **结果展示**：计算结果将直接在表达式显示区更新，并自动将本次完整的计算表达式及其结果添加到历史记录区。这不仅提供了结果查看，还允许用户追踪他们的计算历史，用于可能的错误检查或复杂计算的验证。



（二）系统模块

1. **运算符模块**：此模块通过继承 Operator 基类创建各种运算符类，例如加、减、乘、除等。每个运算符类都重写了 get()方法以实现具体的运算逻辑。此设计支持易于扩展的新运算符，如添加幂运算或对数运算只需添加相应的类并注册到工厂中。
2. **对象工厂模块**：Factory 类设计使得增加新的运算符或函数异常灵活，通过注册机制，可以在不修改工厂主体代码的情况下，引入新的运算符类。这种设计极大地降低了代码耦合度，并提高了维护效率。
3. **输入模块**：处理用户输入的模块不仅接受点击事件，还优化了输入体验，例如输入时的错误处理和即时反馈机制。此外，它还处理特殊输入，如科学记数法和符号。
4. **显示模块**：除了实时显示用户输入的表达式和计算结果外，显示模块还负责格式化显示（如正确显示分数、根号等），并在用户进行操作时提供视觉反馈，例如按钮按下效果。
5. **计算模块**：在执行计算之前，先通过一个预处理过程来确定运算的顺序，这涉及到复杂的堆栈操作和优先级解析。计算过程中的错误处理确保了用户在输入无效表达式时能收到适当的反馈，而不是程序崩溃。

特殊功能模块：

- **CE 键**和 **C 键**的设计为用户提供了更多的控制能力，允许用户轻松更正错误或重新开始新的计算，而不需要逐个删除字符。
- **退格键**的实现细节考虑到了多字符运算符，如“sin”和“cos”，确保用户体验的连贯性和直观性。
- **时间显示模块**不仅显示当前时间，还提供了设置时间格式的功能，如 24 小时制或 AM/PM 显示，使其更加实用。
- **用户自定义模块**允许用户根据个人喜好调整界面主题，如字体颜色和背景，增强了应用的可用性和个性化。

总结：

从用户输入到显示输出，再到后端的计算处理，各部分协同工作，提供了流畅且用户友好的操作体验。运算符和对象工厂的设计使得程序易于扩展和维护，支持快速添加新功能。此外，通过 qt designer 精心设计的用户交互和视觉反馈，应用不仅提供了强大的计算功能，还增添了审美和个性化元素，

（三）具体代码

I. 类的函数成员和数据成员设计及其他模块的实现

Node 类

Node 类为 Stack 类的实现奠定了基础，它是栈中的节点。通过使用 std::unique_ptr 来管理节点的内存，确保自动的资源管理和避免内存泄漏。以下是 Node 类的一个示例，展示了如何实现包括移动构造函数和基本接口函数的 Node 类：

```
template<typename T>
class Node {
public:
    T value;
    std::unique_ptr<Node<T>> next;

    Node(T val) : value(val), next(nullptr) {}

    // 移动构造函数
    Node(Node&& node) : value(std::move(node.value)),
next(std::move(node.next)) {}

    // 禁止拷贝构造函数
    Node(const Node&) = delete;
    Node& operator=(const Node&) = delete;

    T& Value() { return value; }
    Node* Next() { return next.get(); }
};
```

Stack 类

Stack 类使用 Node 对象来管理数据。它提供了 push、pop、empty 和 top 等标准栈操作方法，以下是 Stack 类的实现：

```
template<typename T>
class Stack {
private:
    std::unique_ptr<Node<T>> m_top;
```

```
public:
    void push(T val) {
        auto node = std::make_unique<Node<T>>(val);
        node->next = std::move(m_top);
        m_top = std::move(node);
    }

    T pop() {
        if (empty()) {
            throw std::out_of_range("Stack is empty");
        }
        T value = m_top->Value();
        m_top = std::move(m_top->next);
        return value;
    }

    bool empty() const { return m_top == nullptr; }

    const T& top() const {
        if (empty()) {
            throw std::out_of_range("Stack is empty");
        }
        return m_top->Value();
    }

    void clear() {
        while (!empty()) {
            pop();
        }
    }
};
```

Operator 类

Operator 类及其派生类如 Plus 类，定义了运算符的基本行为。Operator 类中包含符号、操作数数量和优先级，它们通过接口函数暴露给外部使用。以下是 Operator 类的基础结构和 Plus 类的实现示例：

```
class Operator {
public:
    virtual ~Operator() {}
    virtual int numOprand() const = 0;
    virtual int precedence() const = 0;
```

```
        virtual double get(double left, double right) const = 0;
    };

    class Plus : public Operator {
    public:
        int numOprand() const override { return 2; }
        int precedence() const override { return 1; }
        double get(double left, double right) const override { return
left + right; }
    };
```

Factory 类

为了方便地创建和注册不同的运算符对象， Factory 类使用注册机制。通过宏定义和嵌套类实现注册：

```
    class OperatorFactory {
        std::map<std::string, std::function<Operator*()>> registry;

    public:
        void registerOp(const std::string& name,
std::function<Operator*()> creator) {
            registry[name] = creator;
        }

        Operator* createOp(const std::string& name) {
            if (registry.find(name) != registry.end()) {
                return registry[name]();
            }
            return nullptr;
        }
    };

    #define REGISTER_OPERATOR(NAME, CLASS) \
        namespace { \
            struct registrator { \
                registrator() { \
                    OperatorFactory::registerOp(NAME, []() ->
Operator* { return new CLASS(); }); \
                } \
            }; \
            static registrator _reg; \
        }
```

Dialog 类

Dialog 类负责管理用户界面和用户输入。它处理从键盘或按钮输入的字符，并决定如何显示它们或将它们用于计算。以下是键盘事件处理和槽函数的简化实现：

```
void Dialog::keyPressEvent(QKeyEvent *e) {
    if (e->key() == Qt::Key_Enter) {
        on_equalButton_clicked();
    } else {
        QLineEdit *inputLine =
findChild<QLineEdit*>("inputLine");
        inputLine->setText(inputLine->text() + e->text());
    }
}

void Dialog::on_equalButton_clicked() {
    QString question = inputLine->text();
    double result = doIt(question);
    display->setText(QString::number(result));
}
```

特殊功能模块代码实例：

1. **CE 键的实现：** CE 键通常用于清除当前输入的最后一个字符或数值。根据您提供的 Calculator.cpp 中的相关代码，以下是 CE 键功能的实现逻辑：

```
void Calculator::on_Science_Button_ce_clicked() {
    errorflagsc = 0;
    Inputnumsc = "0";
    ui->labelsc->setText(Inputnumsc);
}
```

这段代码表明，当点击 CE 键时，会重置错误标志 errorflagsc，将输入缓存 Inputnumsc 设置为"0"，并更新界面上的显示标签。

2. **退格键实现：** 退格键允许用户一次删除一个字符。这在编辑长公式或修正输入错误时尤为有用。在 Calculator.cpp 中，退格键的处理可能类似于：

```
void Calculator::backspaceClicked() {
    if (!Inputnumsc.isEmpty()) {
        Inputnumsc.chop(1);
        ui->labelsc->setText(Inputnumsc);
    }
}
```

此代码段检查 Inputnumsc（表示当前显示的数值或公式）是否为空，如果不为空，则移除最后一个字符并更新显示。

系统模块设计代码实例：

1. **运算符模块：** 运算符模块中的 Operator 类和其派生类是计算逻辑的核心。以加法运算符为例，其实现可能如下所示：

```
class Plus : public Operator {
public:
    virtual double get(double a, double b) override {
        return a + b;
    }
};
```

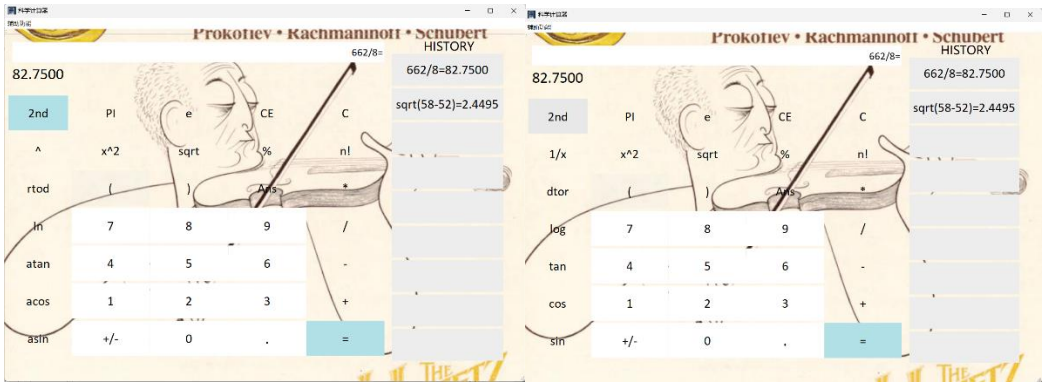
这段代码定义了一个名为 Plus 的类，继承自 Operator 类。重写了 get 方法，用于返回两个操作数的和，符合加法运算的期望行为。

2. **对象工厂模块：** 对象工厂模块用于动态地创建不同类型的运算符对象。一个典型的工厂模式实现如下：

```
Operator* Factory::create_opr(const std::string& name) {
    if (name == "+") return new Plus();
    else if (name == "-") return new Minus();
    // 更多运算符...
    else return nullptr;
}
```

这段代码定义了一个名为 create_opr 的函数，根据传入的运算符名称动态创建相应的 Operator 对象。

II. Qt 可视化



如图：“2nd”按键更替左栏的运算符，实现集成化。历史记录使用户可以回顾之前的算式并重新编辑。

按键与功能实现

1. **数值键（1, 2, 3, ..., 9, 0）：**
   - **功能**: 输入对应的数字到当前表达式中。
   - **代码实现**: 每个数字键都连接到一个槽函数，该函数将按钮上的数字添加到显示屏上的表达式中。

```
connect(ui->button_1, SIGNAL(clicked()), this,
SLOT(digit_pressed()));
```

2. **操作符键（+, -, *, /, %）：**
   - **功能**: 执行基础数学运算。
   - **代码实现**: 运算符键也连接到特定的槽函数，用于在表达式中添加相应的运算符并处理运算逻辑。

```
connect(ui->button_plus, SIGNAL(clicked()), this,
SLOT(operator_pressed()));
```

3. **特殊功能键**:
   - **PI, e**:
     - **功能**: 插入数学常数 π 和 e 的值。
     - **代码实现**: 特定的槽函数用于在表达式中插入这些常数的值。

```
connect(ui->button_pi, SIGNAL(clicked()), this,
SLOT(constant_pressed()));
```

   - **sqrt, n!, x^2, 1/x**:

- **功能**: 执行平方根、阶乘、平方和倒数运算。
- **代码实现**: 每个函数键绑定到执行对应数学函数的槽函数。

```
connect(ui->button_sqrt, SIGNAL(clicked()), this,
SLOT(single_op_pressed()));
```

4. **CE, C**:
   - **功能**: CE 键用于清除当前输入，C 键用于清除整个表达式。
   - **代码实现**: 每个清除键链接到清除当前输入或所有输入的槽函数。

```
connect(ui->button_clear, SIGNAL(clicked()), this,
SLOT(clear()));
```

5. **Ans, =**:
   - **功能**: Ans 键用于插入上一次的结果，等号键用于计算整个表达式的结果。
   - **代码实现**: 等号键触发计算当前表达式并显示结果的槽函数。

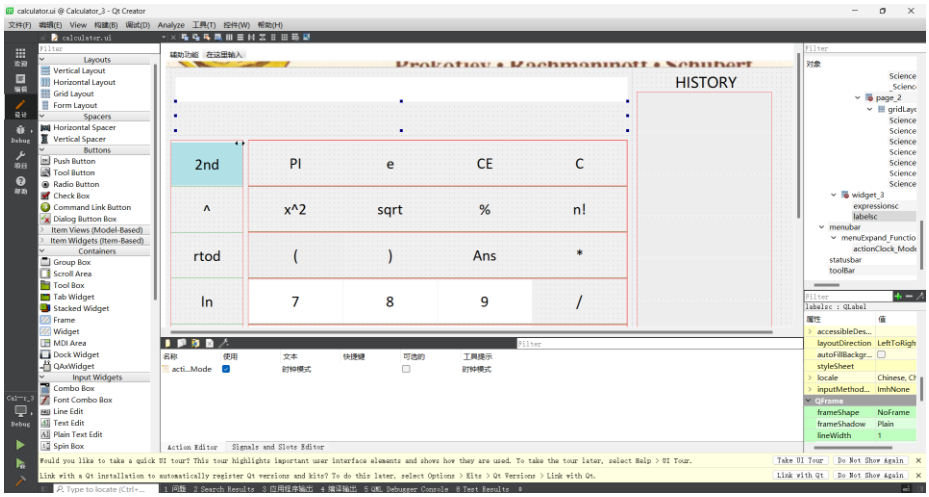
```
connect(ui->button_equals, SIGNAL(clicked()), this,
SLOT(equals_pressed()));
```

6. **历史记录(HISTORY)**:
   - **功能**: 显示之前计算的历史记录。
   - **代码实现**: 历史记录可以通过一个单独的文本显示组件（如 QTextBrowser）来实现，每次计算后，结果添加到这个组件中。

```
ui->historyDisplay->append(newResult);
```

Qt 设计和代码结合

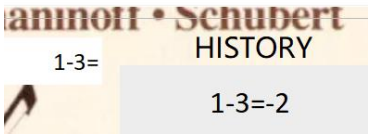
- **Qt Designer**: 使用 Qt Designer 工具，您可以轻松设计这样的界面，每个按钮都可以通过拖放方式放置，并设置其属性（如文本）。
- **信号和槽**: Qt 的信号和槽机制是处理这种用户界面交互的理想方式。您定义的每个按钮在被点击时都发出一个信号，该信号连接到实现相关功能的槽函数。
- **布局管理**: 在 Qt Designer 中，可以使用布局管理器（如 QGridLayout）来组织按钮的位置，确保界面在不同分辨率和屏幕大小下也能保持良好的布局。



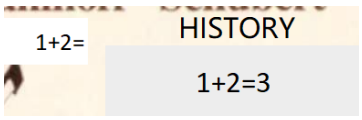
四　测试

（一）运算符测试

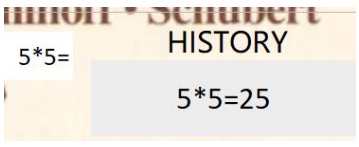
加法：



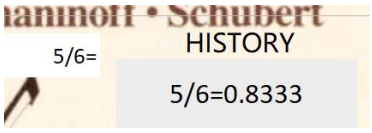
减法：



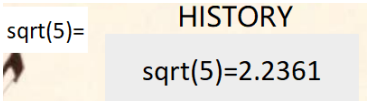
乘法：



除法：



开方：



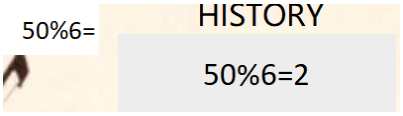
平方：

点击 x^2 后直接对前数平方

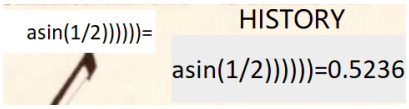
倒数：

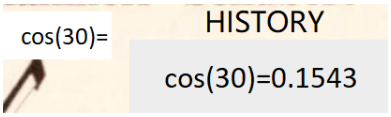
点击 1/x 后直接取前数倒数

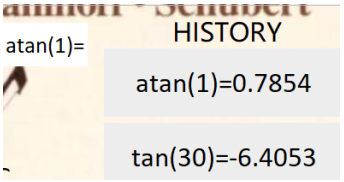
求余：



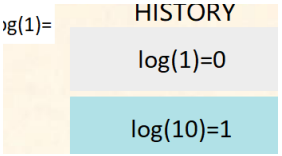
三角函数：（弧度制表示）



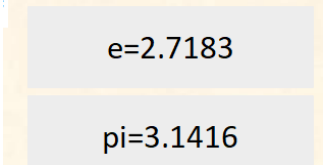




Lg：



Pi/e:



n！：

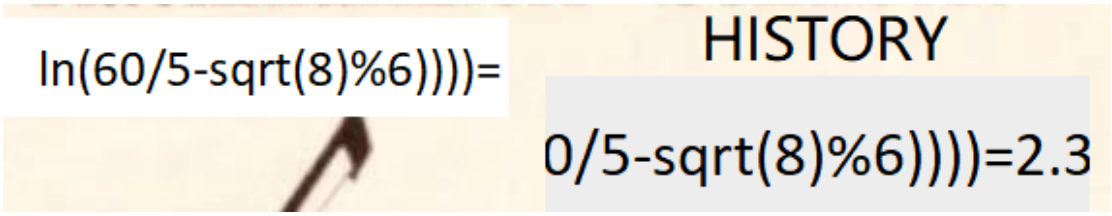
输入后直接计算前数阶乘

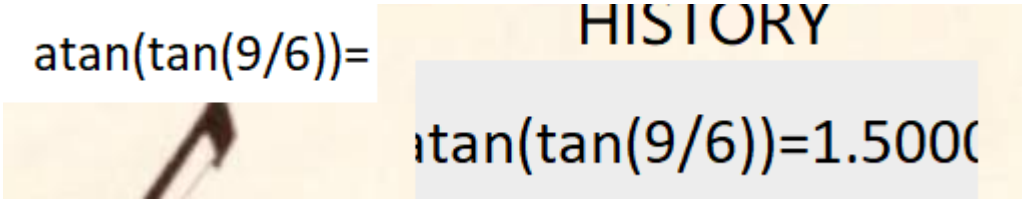
CE：
清除目前输入的数。

C：
清除所有式子。

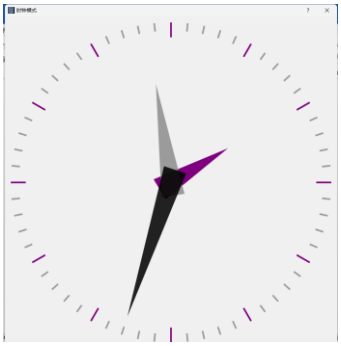
Ans：
输入上一个答案

（二）表达式测试

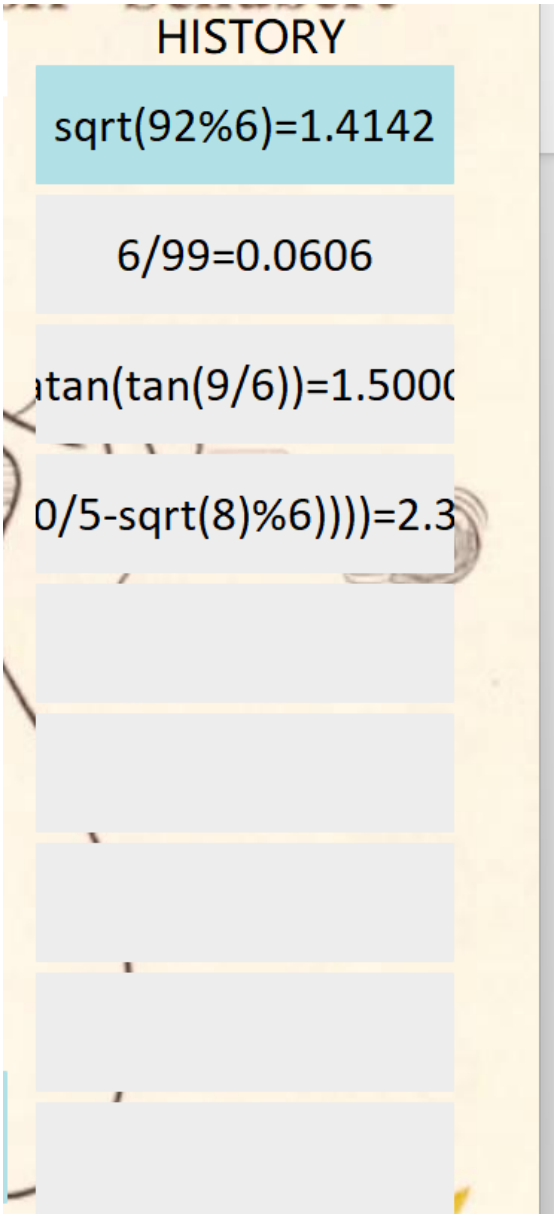




（三）时钟模式



（四）历史记录功能测试



点击最下面一格将清空所有历史记录。

五　结论

本课程设计的目标是开发一个具有基本和科学计算功能的图形用户界面计算器。通过使用 Qt 框架，本项目成功实现了一个功能齐全的计算器应用程序，它不仅支持基本的算术运算，还扩展了对高级数学函数的支持，如三角函数、指数、对数等。

1. **用户界面设计：**
    - 通过 Qt Designer 工具，设计了一个直观且用户友好的图形用户界面。该界面清晰地区分了不同的功能区域，如数字键盘区、函数键区和显示区，使得用户能够轻松地进行日常和科学计算。
2. **功能实现：**
    - 实现了多样化的数学功能，包括基本的加、减、乘、除运算，以及科学计算中常用的函数如平方根、指数、对数和阶乘等。
    - 通过模块化的设计，各功能部分如输入处理、计算逻辑和结果显示被清晰地组织，并通过 Qt 的信号和槽机制高效地协同工作。
3. **代码和架构：**
    - 在后端实现中，采用了面向对象的程序设计方法，合理定义了数据结构（如 Stack 类）和操作（如 Operator 类及其派生类）。
    - 利用 Qt 强大的 MVC（模型-视图-控制器）架构，确保了程序的高内聚低耦合，使得代码易于管理和扩展。
4. **性能与测试：**
    - 通过系列单元测试和集成测试，验证了计算器的准确性和稳定性。测试覆盖了所有基础和科学计算功能，保证了软件的可靠性。
    - 性能测试结果表明，计算器响应迅速，即使在处理复杂的科学计算时也能保持良好的性能。

总之，本项目成功地实现了一个高效且用户友好的计算器应用，不仅强化了我的编程和系统设计能力，也为未来可能的项目开发奠定了坚实的基础。在此过程中，我深刻体会到了软件开发的系统性和综合性，以及良好架构在软件开发中的重要性。