实验一 多彩计算器

1.程序介绍

该程序为名称为多彩计算器，设计的思路为仿照 window10 官方的计算器，用 Qt 框架配合 c++语言来完成。在原有计算器加减乘除的功能下扩展为支持浮点运算，进一步利用栈来解决运算优先级的问题，在页面引入 e，π 等常见的数学字符,还扩展了乘方和进制转换的功能。为了更好的呈现一个美好的 UI 界面，在保证计算器严谨的同时，添加彩色触显，来增强用户体验感。

2.操作说明

2.1 清除和退格操作

这里先输入 20201231，如图 1-1 所示，再进行退格操作（按钮 C），后显示如图 1-2 所示，再进行清空操作（按钮 AC），效果如图 1-3 所示。

图 1 清除和退格展示图

2.2 加减乘除操作

这里为了一次性检验加减乘除，先输入 8*2/4+5-2 如图 2-1 所示，运算结果应该为 7，按下等于检验结果如图 2-2 所示。

图 2 加减乘除运算图

2.3 扩展高级操作

1. 乘方操作

这里输入 5^2 来进行验证，结果如图 3 所示。

图 3 乘方操作

2. 分数操作

这里输入 25 再按（1/x 按钮）来进行验证，结果如图 4 所示。

图 4 分数操作

3. 取余操作

这里输入 25%4 来进行验证，结果如图 5 所示。

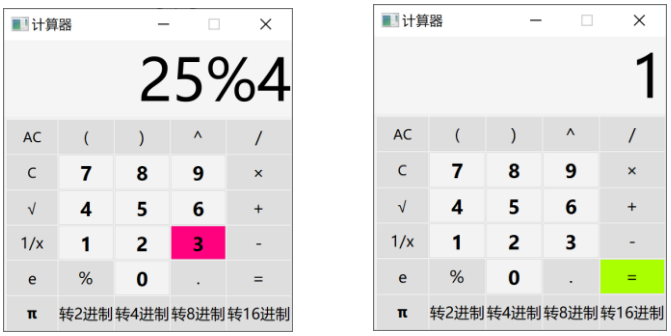

图 5 取余操作

4. 根号操作

这里输入 25 再按（√ 按钮）来进行验证，结果如图 6 所示。

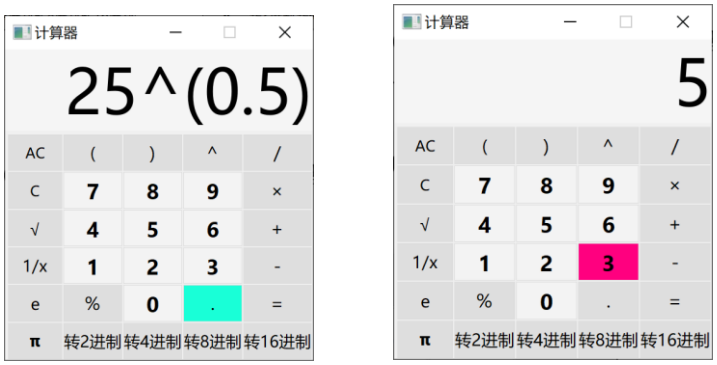

图 6 根号操作

2.4 特色字符展示

1. 圆周率 π

这里输入 π*2 来进行验证，结果如图 7 所示。

图 7 圆周率

2. 自然指数 e

这里输入 e*2 来进行验证，结果如图 8 所示。

图 8 自然指数

2.5 优先级与小数展示

这里输入（2.5+1）*2 来进行验证，结果如图 9 所示。

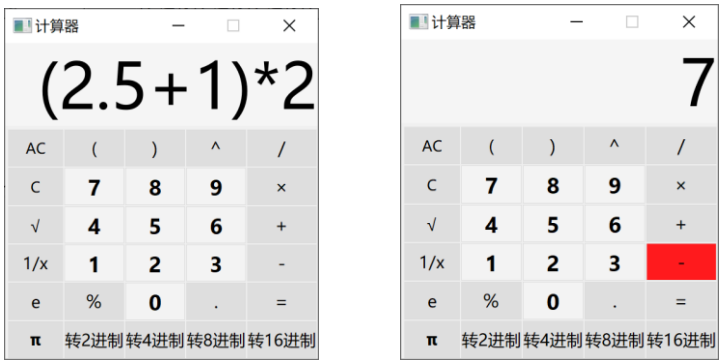

图 9 优先级展示

2.6 进制转换

这里我们输入 10 为例，输出结果如图 10 所示.

图 10 进制转换

3.设计理念

3.1 设计目标

该项目是想设计一个支持连续计算的四则运算计算器，通过点击相应的数字

按钮和运算按钮，输入并完成如 4*6+3、或者取倒数等等类似的计算，并将运算结果显示输出在文本框中，同时此计算器也具备清空、退格等其他功能，页面采用灰白配色，在按动时有不同颜色彩蛋。其运行界面如图 11 所示。

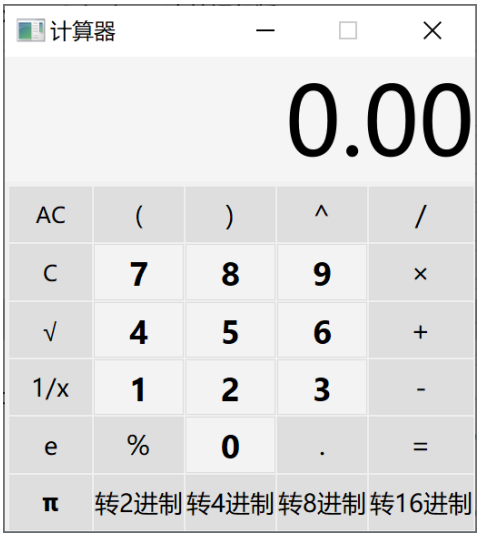

图 11 计算器页面

3.2 设计分析和算法分析

3.2.1 页面搭建

这里采用 Qt 框架自带的 UI 界面进行美化排布，效果如下图所示，利用 UI 自动生成按钮等类，在页面美化中，添加样本样式，利用 Qt 自带的 QSS 进行美化（操作如前端 CSS 类似）。

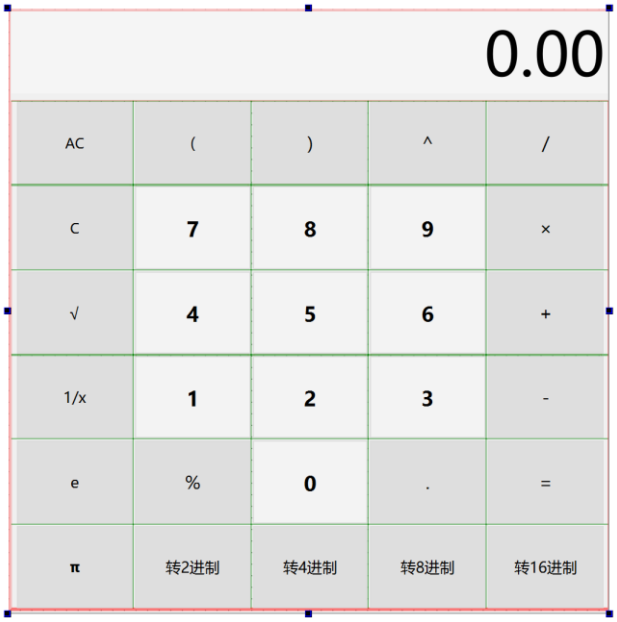

图 12UI 界面美化

3.2.2 按钮绑定

利用 UI 生成的类，添加槽函数进行关联绑定，生成类如图 13 所示，在 mainwindow.cpp 析构函数中进行绑定操作。代码如下：

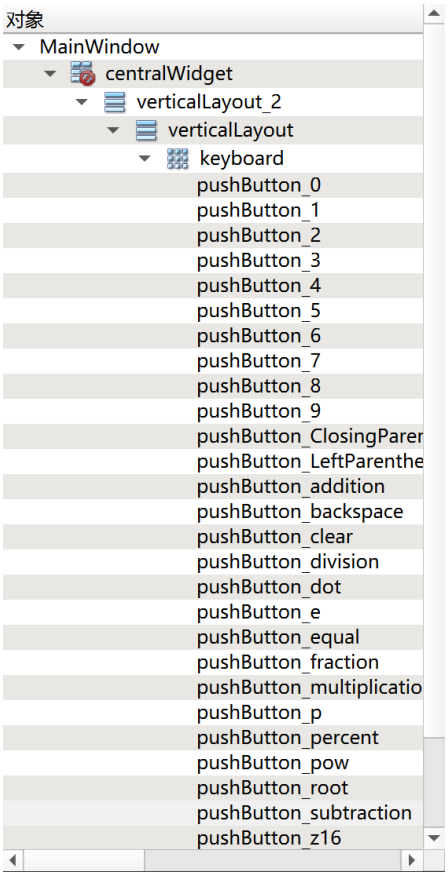

图 13 按钮类

```
    ui->setupUi(this);
    //清空a，b
    a.clear();
    b.clear();

    //绑定按键0 与处理函数
        connect(ui->pushButton_0,&QPushButton::clicked,[=](){btn_logic(0);});
        //绑定按键1 与处理函数
        connect(ui->pushButton_1,&QPushButton::clicked,[=](){btn_logic(1);});
        //绑定按键2 与处理函数
        connect(ui->pushButton_2,&QPushButton::clicked,[=](){btn_logic(2);});
        //绑定按键3 与处理函数
        connect(ui->pushButton_3,&QPushButton::clicked,[=](){btn_logic(3);});
        //绑定按键4 与处理函数
        connect(ui->pushButton_4,&QPushButton::clicked,[=](){btn_logic(4);});
        //绑定按键5 与处理函数
        connect(ui->pushButton_5,&QPushButton::clicked,[=](){btn_logic(5);});
        //绑定按键6 与处理函数
        connect(ui->pushButton_6,&QPushButton::clicked,[=](){btn_logic(6);});
        //绑定按键7 与处理函数
        connect(ui->pushButton_7,&QPushButton::clicked,[=](){btn_logic(7);});
        //绑定按键8 与处理函数
        connect(ui->pushButton_8,&QPushButton::clicked,[=](){btn_logic(8);});
        //绑定按键9 与处理函数
        connect(ui->pushButton_9,&QPushButton::clicked,[=](){btn_logic(9);});
        //绑定按键点与处理函数
        connect(ui->pushButton_dot,&QPushButton::clicked,[=](){btn_logic(0,".");});

        //绑定按键+与处理函数
        connect(ui->pushButton_addition,&QPushButton::clicked,[=](){btn_logic(0,"+");});
        //绑定按键-与处理函数
        connect(ui->pushButton_subtraction,&QPushButton::clicked,[=](){btn_logic(0,"-
");});
        //绑定按键*与处理函数

connect(ui->pushButton_multiplication,&QPushButton::clicked,[=](){btn_logic(0,"*");});
        //绑定按键/与处理函数
        connect(ui->pushButton_division,&QPushButton::clicked,[=](){btn_logic(0,"/");});
        //绑定按键%与处理函数
        connect(ui->pushButton_percent,&QPushButton::clicked,[=](){btn_logic(0,"%");});
        //绑定按键^与处理函数
```

```
        connect(ui->pushButton_pow,&QPushButton::clicked,[=](){btn_logic(0,"^");});

        //绑定按键(与处理函数

connect(ui->pushButton_LeftParenthesis,&QPushButton::clicked,[=](){btn_logic(0,"(");});
        //绑定按键)与处理函数

connect(ui->pushButton_ClosingParenthesis,&QPushButton::clicked,[=](){btn_logic(0,")");});

        //绑定按键 e 与处理函数
        connect(ui->pushButton_e,&QPushButton::clicked,[=](){btn_logic(0,"e");});
        //绑定按键 p 与处理函数
        connect(ui->pushButton_p,&QPushButton::clicked,[=](){btn_logic(0,"p");});
        //绑定按键根号与处理函数
        connect(ui->pushButton_root,&QPushButton::clicked,[=](){btn_logic(0,"root");});
        //绑定按键 1/x 与处理函数

connect(ui->pushButton_fraction,&QPushButton::clicked,[=](){btn_logic(0,"pushButton_fracti
on");});

        //绑定按键 AC 与处理函数
        connect(ui->pushButton_clear,&QPushButton::clicked,[=](){
            a.clear();
            chh.clear();
            b.clear();
            ui->lineEdit->setText(a);
        });

        //绑定按键退格-> 与处理函数
        connect(ui->pushButton_backspace,&QPushButton::clicked,[=](){
            //删除 a.pop
            a.chop(1);
            chh.chop(1);
            ui->lineEdit->setText(a);
        });

        //绑定按键转 2 进制与处理函数
        connect(ui->pushButton_z2,&QPushButton::clicked,[=](){

            int number = a.toInt();
            a.clear();
            decimalToAny(number,2);
```

```
        ui->lineEdit->setText(a);
    });
    //绑定按键转 4 进制与处理函数
    connect(ui->pushButton_z4,&QPushButton::clicked,[=](){

        int number = a.toInt();
        a.clear();
        decimalToAny(number,4);
        ui->lineEdit->setText(a);
    });
    //绑定按键转 8 进制与处理函数
    connect(ui->pushButton_z8,&QPushButton::clicked,[=](){

        int number = a.toInt();
        a.clear();
        decimalToAny(number,8);
        ui->lineEdit->setText(a);
    });
    //绑定按键转 16 进制与处理函数
    connect(ui->pushButton_z16,&QPushButton::clicked,[=](){

        int number = a.toInt();
        a.clear();
        decimalToAny(number,16);
        ui->lineEdit->setText(a);
    });


    //绑定按键= 与处理函数
    connect(ui->pushButton_equal,&QPushButton::clicked,[=](){den_logic();});
```

代码块 1 析构函数代码

2.2.3 进制转换算法

构建 cimalToAny 函数，输入转换的 10 进制数 n，和转换进制 d，利用辗转相除法求余数来构建。例如以 2 进制为例

除 2 取余法，即每次将整数部分除以 2，余数为该位权上的数，而商继续除以 2，余数又为上一个位权上的数，这个步骤一直持续下去，直到商为 0 为止，最后读数时候，从最后一个余数读起，一直到最前面的一个余数。算法代码如下：

```
void MainWindow::decimalToAny(int n,int d)//10进制转换成任意进制
{
    if(n==0)
        return ;
    else
    {
        decimalToAny(n/d,d);
        if(d>=10)//如果是10进制以上
        {
            if(n%d>=10)
            {
                 //printf("%c",(char)((n%d-10)+'A'));
                a += char((n%d-10)+'A');
            }
            else//如果余数小于10，则直接输出
            {
                //printf("%d",n%d);
                a += QString::number(n%d);
            }
        }
        //如果进制小于10，不会有字母的问题
        else
        {
            //printf("%d",n%d);
            a += QString::number(n%d);
        }

    }
}
```

代码块 2 进制转换代码

2.2.4 字符转换

根据按钮调用此函数 btn_logic,对与屏幕进行输入,输入类型为 QString 类,改变输入并刷新屏幕。代码如下:

```
void MainWindow::btn_logic(int x , QString i)
{
    if(i != " " )
    {
        if(i == "pushButton_fraction")
        {
            a += "^(-1)";
            chh += "^(-1)";
        }
        else if (i == "root")
        {
            a += "^(-1)";
            chh += "^(-1)";
        }
        else if (i == "p")
        {
            a += "π";
            chh += "p";
        }
        else
        {
            a += i;
            chh += i;
        }
    }
    else
    {
        a += QString::number(x);
        chh += QString::number(x);
    }
    ui->lineEdit->setText(a);

}
```

代码块 3btn_logic 函数代码

2.2.5 逻辑运算代码

这里采用面向对象的方法，构建一个运算类 calc 和检验类 check，通过构建俩个栈，一个操作数一个运算符，将输入的中序运算改为后序运算并计算结果输出，下代码块相关实现：

```
#ifndef CLAC_H
#define CLAC_H
#include <iostream>
#include <fstream>
#include <stack>
#include <cctype>
#include <sstream>
#include <iomanip>
#include <cmath>

using namespace std;

class calc
{
private:
    stack<char> operators;
    stack<double> operands;

    char getSign(void)
    {
        return operators.top();
    }

    double getNum(void)
    {
        return operands.top();
    }

    void popSign(void)
    {
        operators.pop();
    }

    void popNum(void)
    {
        operands.pop();
    }

    double popAndGetNum(void)
    {
        double num = getNum();
        popNum();
```

```
        return num;
    }

    char popAndGetSign(void)
    {
        char sign = getSign();
        popSign();
        return sign;
    }
public:
    double final_result;

    void push(double num)
    {
        operands.push(num);
    }

    void push(char sign)
    {
        operators.push(sign);
    }

    char get(void)
    {
        return getSign();
    }

    void pop(void)
    {
        popSign();
    }

    double result(void)
    {
        if (!operands.empty())
            return getNum();
        return 0.0;
    }

    void calculate(void)
    {
        double post = popAndGetNum();
```

```
        char sign = popAndGetSign();
        double pre = popAndGetNum();
        double result = 0.0;
        switch (sign)
        {
        case '+':
            result = pre + post;
            break;
        case '-':
            result = pre - post;
            break;
        case '*':
            result = pre * post;
            break;
        case '/':
            if (fabs(post) < 1e-6)
            {
                calc::writeResult("#1");
                exit(EXIT_SUCCESS);
            }
            else
                result = pre / post;
            break;
        case '^':
            result = pow(pre, post);
            break;
        case '%':
            result = static_cast<int>(pre) % static_cast<int>(post);
            break;
        }
        //cout << result << endl;
        final_result = result;
        push(result);
    }

    bool canCalculate(char sign)
    {
        if (sign == '(' || sign == '[' || sign == '{' || operators.empty())
            return false;
        char t = getSign();
        if (t == '^')
            return true;
```

```
        switch (t)
        {
        case '+':
        case '-':
            return sign == '+' || sign == '-';
        case '*':
        case '/':
        case '%':
            return sign == '+' || sign == '-' || sign == '*' || sign == '/'
|| sign == '%';
        }
        return false;
    }

    bool empty(void)
    {
        return operators.empty();
    }

    static void writeResult(const string& s)
    {
        ofstream out;
        out.open("result.txt");
        out << s;
        out.close();
    }

    void writeResult(void)
    {
        ofstream out;
        out.open("result.txt");
        if (result() < 1e15)
        {
            out << fixed;
            out << setprecision(12);
        }
        out << result();
        out.close();
    }
};
```

```
class check
{
private:
    string s;
    size_t len;
    char c;
    bool valid(void)
    {
        if (isSignOrDot(0) || isRightBrace(0))
            return false;
        len = s.size();
        stack<char> braces;
        for (size_t i = 0; i < len; ++i)
        {
            if (isLeftBrace(i))
            {
                if (isSignOrDot(i + 1))
                {
                    if (s[i + 1] != '-' && s[i + 1] != '+')
                        return false;
                }
                if (isRightBrace(i + 1))
                    return false;
                braces.push(s[i]);
            }
            else if (isRightBrace(i))
            {
                if (isDot(i + 1) || isDigit(i + 1) || isLeftBrace(i + 1))
                    return false;
                if (isRightBrace(i + 1))
                {
                    stack<char> braces_copy(braces);
                    if (braces_copy.empty())
                        return false;
                    braces_copy.pop();
                    if (braces_copy.empty())
                        return false;
                }
                if (braces.empty())
```

```
                    return false;
                char brace = braces.top();
                if ((brace == '(' && s[i] != ')') || (brace == '[' &&
s[i] != ']') || (brace == '{' && s[i] != '}'))
                    return false;
                braces.pop();
            }
            else if (isSign(i))
            {
                if (isSign(i + 1) || isDot(i + 1) || isRightBrace(i + 1))
                    return false;
            }
            else if (isDot(i))
            {
                if (isSignOrDot(i + 1) || isBrace(i + 1))
                    return false;
            }
            else if (isDigit(i))
            {
                if (isRightBrace(i + 1))
                {
                    if (braces.empty())
                        return false;
                    char brace = braces.top();
                    if ((brace == '(' && s[i + 1] != ')') || (brace == '['
&& s[i + 1] != ']') || (brace == '{' && s[i + 1] != '}'))
                        return false;
                }
            }
        }
        return braces.empty();
    }

    bool set(size_t i)
    {
        if (i >= len)
            return false;
        c = s[i];
        return true;
    }

    bool isSign(size_t i)
```

```
    {
        if (set(i))
            return c == '+' || c == '-' || c == '*' || c == '/' || c == '^'
|| c == '%';
        return false;
    }

    bool isDot(size_t i)
    {
        if (set(i))
            return c == '.';
        return false;
    }

    bool isBrace(size_t i)
    {
        return isLeftBrace(i) || isRightBrace(i);
    }

    bool isLeftBrace(size_t i)
    {
        if (set(i))
            return c == '(' || c == '[' || c == '{';
        return false;
    }

    bool isRightBrace(size_t i)
    {
        if (set(i))
            return c == ')' || c == ']' || c == '}';
        return false;
    }

    bool isSignOrDot(size_t i)
    {
        return isSign(i) || isDot(i);
    }

    bool isDigit(size_t i)
    {
        if (set(i))
            return isdigit(c);
```

```
        return false;
    }

public:

    check() {}
    bool valid(const string& s)
    {
        this->s = s;
        return valid();
    }

    string getResult(void)
    {
        size_t len = s.size();
        for (size_t i = 0; i < len; ++i)
        {
            if (s[i] == '(' && (s[i + 1] == '-' || s[i + 1] == '+'))
                s.insert(i + 1, "0");
        }
        return s;
    }
};









    #endif // CLAC_H
```

代码块 4 运算类代码

2.2.6 等于函数

在按下等于后执行这个 den_logic 函数，将 QString 类型转换为 string 类，并将运算的字符串进行分割，利用 isdigit（）函数来进行有效的判断分割 double 类数字，将运算符压入运算符栈中，并进行栈内运算，调用逻辑运算类进行运算，输入给 QString 并刷新显示。代码如下：

```
void MainWindow::den_logic()
{
    string s = chh.toStdString();
    a.clear();
    check chk;
    if (chk.valid(s))
        s = chk.getResult();
    else
    {
        calc::writeResult("#");
        a += "输入错误";
    }

    size_t len = s.size();
    calc c;
    for (size_t i = 0; i < len; ++i)
    {
        if (isdigit(s[i]))
        {
            double num;
            size_t i1 = i + 1;
            while (i1 < len && (isdigit(s[i1]) || s[i1] == '.'))
                ++i1;
            istringstream input(s.substr(i, i1));
            input >> num;
            i = i1 - 1;
            c.push(num);
        }
        else if (s[i] == '}' || s[i] == ']' || s[i] == ')')
        {
            char sign;
            char start = (s[i] == '}' ? '{' : (s[i] == ']' ? '[' : '('));
            while ((sign = c.get()) != start)
                c.calculate();
            c.pop();
        }
        else if(s[i] == 'p' || s[i] == 'e')
        {
            double number;
            if(s[i] == 'e')
            {
                number = 2.71828182846;
```

```
                c.push(number);
            }
            else
            {
                number = 3.14159265;
                c.push(number);
            }
        }
        else                            //s[i] is [ ( { + - * / ^ %
        {
            while (c.canCalculate(s[i]))
                c.calculate();
            c.push(s[i]);
        }
    }
    while (!c.empty())
    {
        c.calculate();
    }
    a += QString::number(c.final_result);

    ui->lineEdit->setText(a);
}
```

3.3 类图关系

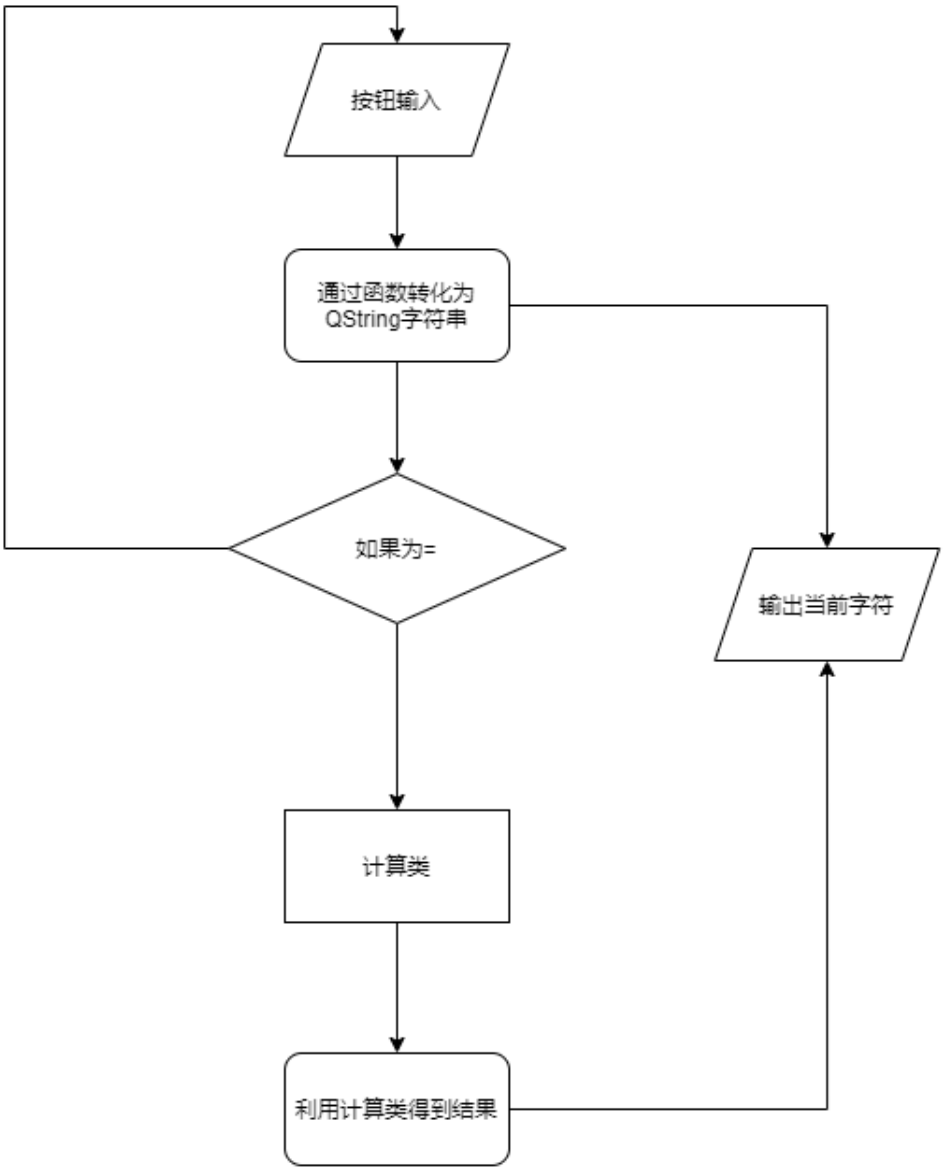

图 14 类关系流程图

4.程序展示

一些操作实际展示，展示效果如下图：

图 15 成果展示图

5.总结思考

通过使用 Qt 应用框架实现了人机交互界面的计算器，采用 Qt 信号槽机制实现计算器的加减乘除及带括号的四则混合运算功能。该程序是一个集继承、图形界面、事件处理等面向对象编程知识的综合应用的实例程序。