

面向对象程序设计大作业

Qt 可视化类模板应用——多项式计算器

学生姓名：杨磊

学 号：2022040176

专 业：计算机科学与技术

班 级：计科 2201

指导教师：江志英

一、功能简介

此次课程大作业完成的多项式计算器的功能涵盖基础的四则运算（其实没有除法运算）和定积分运算。

输入：

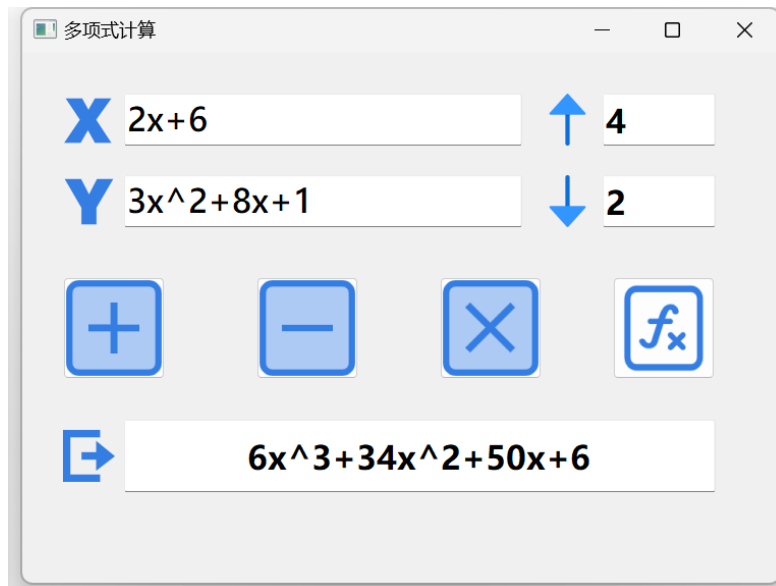
X, Y 为两个多项式。

up 为多项式积分上限顶，down 为定积分下限。

输出：

输出的结果为 output 变量。每次运算都会重置 output 的数值。最终运算的结果由输入多项式 X, Y 和选择的运算方式决定：

1. 加法：在 Algorithm 函数传参时 mode=1。
2. 减法：在 Algorithm 函数传参时 mode=2。
3. 乘法：在 Algorithm 函数传参时 mode=3。
4. 定积分（积分区间由 up, down 输入设置）：在 Algorithm 函数传参时 mode=4；在 numericalIntegration 函数传参时传入上界 up 与下界 down。



二、GUI 界面设计

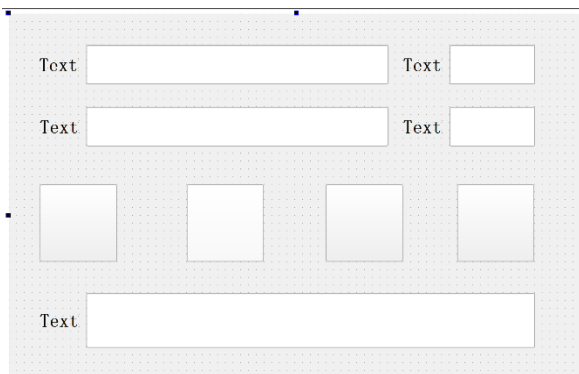
我此次选的 GUI 库为 Qt。

此次界面 UI 设置得较为简洁。用到三种简单的组件：

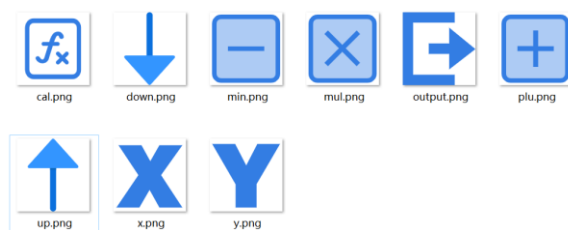
1. Line Edit（用于设置输入变量与打印输出结果）
2. Label（用于展示案件图标）
3. Push Button（用于选择运算方式）

具体设计过程：

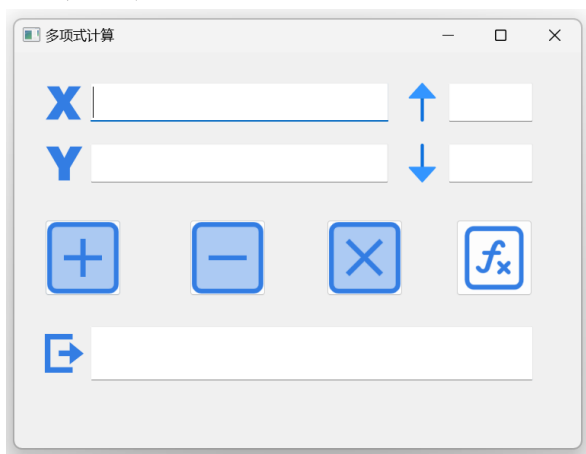
1. 排版拖拽组件按钮到 widget.ui 文件：



2. 添加矢量图形（iconfont-阿里巴巴矢量图标库）：



3. 最终界面：



三、类模板的设计思路与实现过程：

1. 类模板 `polynomial` 定义了多项式的结构，包括系数、指数和指向下一项的指针。

```
template<typename T>
struct polynomial {
    T Coefficient;
    int index;
    polynomial* next;
};
```

2. `calculator` 是主要的计算器类 它包含了多项式的基本操作，如插入、合并、排序、删除等。此外，还有一些其他的功能，如检查输入格式、获取多项式链表、进行加减乘等运算，以及进行定积分运算。

```
class calculator {
public:
    calculator();
    //检查输入格式是否符合规范
    bool check(QString strr);

    //获取每项系数
    double GetCoefficient(char *str);

    //每项输入进入多项式链表中
    template<typename T>
    void push(polynomial<T> *head, T Coefficient, int index);

    //最核心的函数：将输入字符转化为链式的数据结构
    QString GetPolynomialList(QString a, int mp);

    //合并同类型
    template<typename T>
    void CombineSort(polynomial<T> *head);

    //删除指定系数和指数的项
    template<typename T>
    void del(polynomial<T> *DLList1, double Coefficient, int index);

    //升幂排序
    template<typename T>
    void sort(polynomial<T> *DLList1);

    //反转链表
    template<typename T>
    void Inverselist(polynomial<T> *DLList1);

    //将链表末端置空
    template<typename T>
    void Q(polynomial<T> *Ac);

    //定积分函数
    template<typename T>
    double numericalIntegration(polynomial<T> *poly, int down, int up);

    //选择运算方式
    template<typename T>
    QString Algorithm(int CD);

    //格式化输出运算结果
    template<typename T>
    QString Print(polynomial<T> *abc);
```

3. 函数具体实现思路:

①界面输入字符串转化为可存储的整型与浮点型数据存储

//返回系数

double calculator::GetCoefficient(char *str) { ... }

这个函数的主要思路是通过迭代遍历字符串，根据不同的情况提取系数，最终返回提取出的系数值。

初始化了一个布尔变量 falg 用于记录数字的正负。

处理 x 之外的情况：如果当前字符为数字，则开始提取系数。先处理整数部分，将其转换为浮点数存储在 s 中；如果遇到小数点，则跳过，处理小数部分。继续提取小数部分，将其转换为浮点数加到 s 中。

```
if(*str=='+'&&*(str+1)>='0'&&*(str+1)<='9')str++;
if(!(*str>='0'&&*str<='9'))//如果一开始非数字则退出，返回0.0
    return s;
while(*str>='0'&&*str<='9'&&*str!='.')//计算小数点前整数部分
{
    s=s*10.0+*str-'0';
    str++;
}
if(*str=='.')//以后为小数部分
    str++;
while(*str>='0'&&*str<='9')//计算小数部分
{
    s=s+(*str-'0')/d;
    d*=10.0;
    str++;
}
return s*(falg?-1.0:1.0);
```

②将转化后的数据存储到链表结构

QString calculator::GetPolynomialList(QString str , int choose) { ... }

这个函数通过解析输入的多项式字符串，提取系数和指数，并将它们存储到链表中，最后返回处理后的多项式字符串。

调用 push 函数将提取出的系数和指数存储到链表 Aa 或 Bb 中。在存储完所有项之后，调用 CombineSort 函数对多项式链表按照指数的大小排序。返回处理后的多项式：

调用 Print 函数将链表中的内容转换为字符串，并返回处理后的多项式。

```
while(*str+i)
{
    Coefficient=GetCoefficient(str+i);
    if(*str+i!='x')
        i++;
    while((*str+i)>='0'&&*(str+i)<='9')||(*str+i)=='.')
        i++;
    if(*str+i=='+'||*str+i=='-'||*str+i=='\0')
        index=0;
    else
    {
        if(*str+i=='x')
        {
            i++;
            if(*str+i=='+'||*str+i=='-'||*str+i=='\0')
                index=1;
            else
            {
                if(*str+i=='^')
                {
                    i++;
                    index=(int)GetCoefficient(str+i);
                    while((*str+i)>='0'&&*(str+i)<='9')||(*str+i)=='.')
                        i++;
                }
            }
        }
        push(Bb,Coefficient,index);
        CombineSort(Bb);
    }
}
return Print(Bb);
```

③根据 mode 参数选择运算方法

mode=1 加法：遍历两个多项式链表，将对应指数的系数相加，最后合并同类项并排序，得到结果多项式。

mode=2 减法：类似加法，不过第二个多项式的系数取相反数再相加。

mode=3 乘法：双重循环遍历两个多项式链表，将每对系数相乘并加入结果多项式，注意指数相加。

mode=4 定积分：利用数值积分方法，对给定多项式进行定积分计算，返回积分结果。

```
template<typename T>
QString calculator::Algorithm(int mode){
    polynomial<T> *a=Aa;
    polynomial<T> *b=Bb;
    polynomial<T> *c=Cc;
    QString str;
    Q(Cc);
    if(mode==1){//+
    {
        while(a->next){push(c,a->next->Coefficient,a->next->index);a=a->next;}
        while(b->next){push(c,b->next->Coefficient,b->next->index);b=b->next;}
        CombineSort(c);
        str= Print(c);
    }
    else if(mode==2){//-
    {
        while(a->next){push(c,a->next->Coefficient,a->next->index);a=a->next;}
        while(b->next){push(c,-(b->next->Coefficient),b->next->index);b=b->next;}
        CombineSort(c);
        str= Print(c);
    }
    else if (mode == 3) { // *
        polynomial<T> *p = a->next;
        polynomial<T> *q = b->next;
        while (p) {
            while (q) {
                push(c, p->Coefficient * q->Coefficient, p->index + q->index);
                q = q->next;
            }
            p = p->next;
            q = b->next;
        }
        CombineSort(c);
        str = Print(c);
    }
    else if (mode == 4) { // 定积分运算
        T integral_result = numericalIntegration(a, down, up);
        str = QString::number(integral_result);
    }
    return str;
}
}
```

④梯形面积进行积分运算

将积分区间划分成多个小区间，并在每个小区间上使用梯形面积近似计算积分值。在循环中，对于多项式的每一项，通过步长逐步逼近积分区间，计算每个小区间上的梯形面积并累加。最后求和。

```
template<typename T>
double calculator::numericalIntegration(polynomial<T> *poly, int down, int up) {
    polynomial<T> *p = poly->next;
    T result = 0;
    // 设置步长
    const T step = 0.00001;

    while (p) {
        T term_integral = 0;
        for (T x = down; x < up; x += step) {
            term_integral += step * (p->Coefficient * pow(x, p->index) + p->Coefficient * pow(x + step, p->index)) / 2;
        }
        result += term_integral;
        p = p->next;
    }
    return result;
}
}
```

其中传入的参数 poly 指向多项式链表的指针；down：这是积分的下限。up：这是积分的上限，即积分区间的结束值。

四、可视化程序的设计思路与实现过程：

可视化程序使用了 Qt 库，采用了简单的窗口界面设计。

通过四个 Line Edit 组件分别输入了多项式 X，多项式 Y，上界 up，下界 down 的数值。

通过四个 Push Button 组件选择何种模式进行运算处理。

通过一个 Line Edit 组件做为输出 output 输出最终的结果(加法，减法，乘法，定积分共用一个)。

```
//窗口代码
Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);

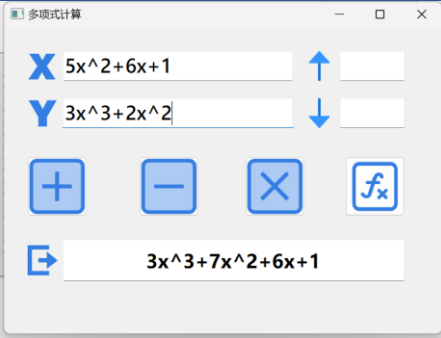
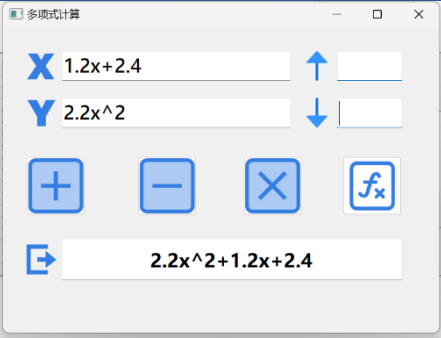
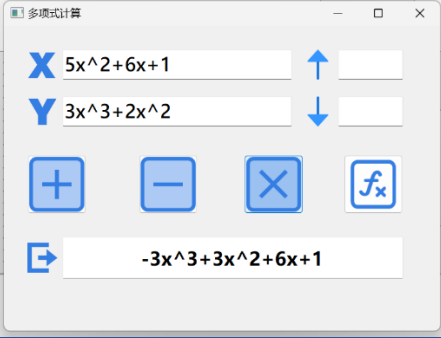
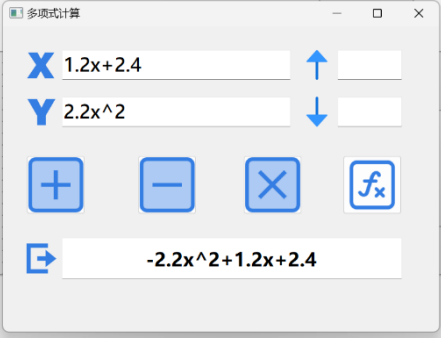
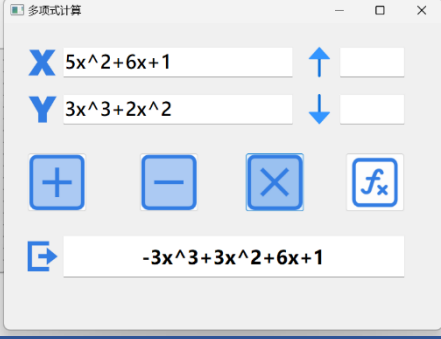
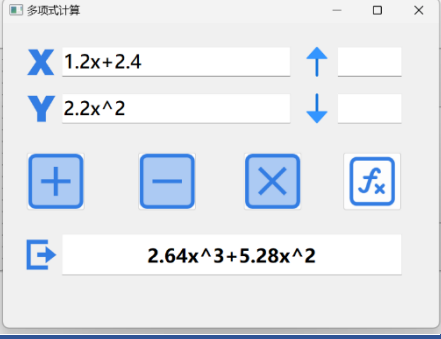
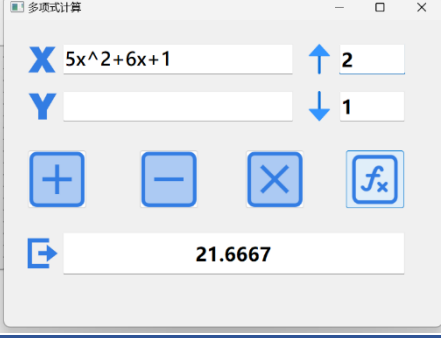
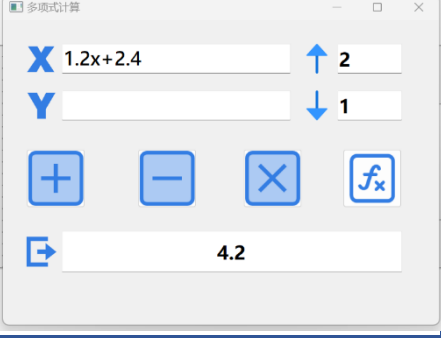
    // 加载图标文件
    QIcon icon1("E:\\桌面\\CS\\面向对象\\大作业\\x.jpg");
    // 设置图标到 QLabel
    ui->label1->setPixmap(icon1.pixmap(QSize(51, 51))); // 设置图标大小
    QIcon icon2("E:\\桌面\\CS\\面向对象\\大作业\\y.jpg");
    ui->label2->setPixmap(icon2.pixmap(QSize(51, 51)));
    QIcon icon3("E:\\桌面\\CS\\面向对象\\大作业\\up.jpg");
    ui->label3->setPixmap(icon3.pixmap(QSize(51, 51)));
    QIcon icon4("E:\\桌面\\CS\\面向对象\\大作业\\down.jpg");
    ui->label4->setPixmap(icon4.pixmap(QSize(51, 51)));
    QIcon icon5("E:\\桌面\\CS\\面向对象\\大作业\\plu.jpg");
    ui->plus->setIcon(icon5);
    ui->plus->setIconSize(QSize(100, 100));
    QIcon icon6("E:\\桌面\\CS\\面向对象\\大作业\\min.jpg");
    ui->minus->setIcon(icon6);
    ui->minus->setIconSize(QSize(100, 100));
    QIcon icon7("E:\\桌面\\CS\\面向对象\\大作业\\mul.jpg");
    ui->multiply->setIcon(icon7);
    ui->multiply->setIconSize(QSize(100, 100));
    QIcon icon8("E:\\桌面\\CS\\面向对象\\大作业\\cal.jpg");
    ui->calculus->setIcon(icon8);
    ui->calculus->setIconSize(QSize(100, 100));

}

Widget::~Widget()
{
    delete ui;
}
```

五、程序的运行截图及结果分析：

程序的多态性体现在 polynomial 类：多项式中每个单项的系数数据类型可以灵活多变。以 double 和 int 类型的系数作为举例。运行截图如下：

	int	double
加法		
减法		
乘法		
定积分		

六、遇到的问题及解决方案：

1. 许多字符串操作在 Qt 环境下不通用

- ① 从 ui 界面传入程序的字符串类型不是普通的 string 类型，而是 Qt 特有的 QString 类型。因此需要做一步数据类型转化，将代码转化为 C 语言风格方便处理：

```
bool calculator::xxx(QString strr)
{
    char str[MAX];
    string strA = string(strr.toLocal8Bit());
    strcpy(str, strA.c_str());
}
```

- ② QString 中的静态成员函数在 output 输出栏中，需要将数据结果转化为 QString 类型。此时 c++ 中常用的 to_string 方法就不再适用了。需要用 QString 中的静态成员函数 QString::number 进行处理。例如：

```
s += QString::number(p->Coefficient);
```

2. 因模板类导致无法解析的外部符号

undefined reference to the function

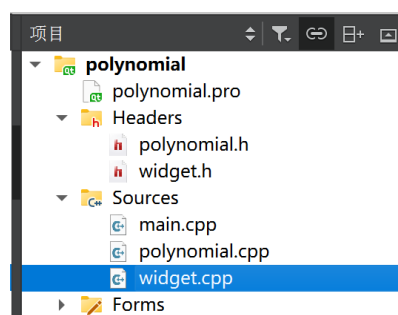
calculator::quick<double>(polynomial<double>*)

问题分析：

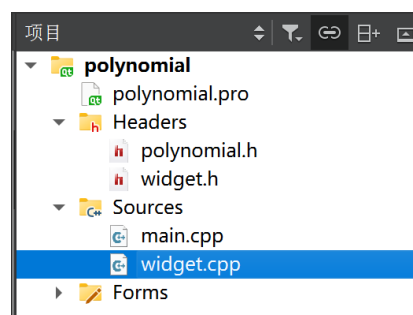
这个问题自己查找了很久都没解决，改了两个小时…本项目一共有两个编译单元，一个是 main，一个是 polynomial，而编译过程是每个编译单元单独编译过后再交给连接器进行连接。在连接器之前两个编译单元就已经进行了编译，而在主函数 main 里面调用外部编译单元时，由于另外一个编译单元 polynomial 在编译时模板类没有被调用而没有得到具现化，从而导致了连接器在函数主函数里调用了模板类函数，但是找不到具体的实现方法的情况，从而就出现了上述报错。

解决方法：

将 polynomial.cpp 中 polynomial 类的具体实现方法转移到 widge 类实现方法这个 widget.cpp 文件中。在编译时直接编译主函数和 widget.h 文件。将两个编译单元融合为一个编译单元。



修改前



修改后

七、总结与反思：

在完成面向对象程序设计大作业的过程是一次深刻的 Debug 体验。

一开始，我选择 Qt 来构建一个可视化的多项式计算器，Qt 库有强大功能和丰富的组件通过 Line Edit、Label 和 Push Button 等组件，我设计了一个简洁而直观的用户界面，让用户能够轻松地进行多项式的基本运算和定积分运算。

在编译运行的过程中遇到超级多的报错比如，Qt 特有的 QString 类型与普通 string 之间的转换问题，以及模板类导致的链接错误等等。花了不少时间与精力去学习和 Debug 这些经历让我对 C++ 语言特性和 Qt 框架有了更深入的理解。

当我看到计算器能够正确地执行加法、减法、乘法和定积分运算时，我开始更多地考虑用户交互的便捷性和界面的直观性，同时也要提高代码的可读性和可维护性。我认为我目前的代码可维护性还欠佳。一个文件写出来大几百行。来回查找函数比较费劲。后续再做类似设计时会注重考虑分离头文件和代码文件。提高工程可读性。