我将为您完善计算器的Qt代码，并解决原有逻辑问题。以下是完整的解决方案，包含详细注释：

### 主要改进：

1. \*\*运算逻辑\*\*：实现先乘除后加减的优先级

2. \*\*表达式处理\*\*：支持多运算符表达式计算

3. \*\*显示逻辑\*\*：操作数输入时不立即计算，按等号才计算

4. \*\*错误处理\*\*：除零错误检测

5. \*\*界面优化\*\*：移除小数点按钮

### calculator.h

```cpp

#ifndef CALCULATOR\_H

#define CALCULATOR\_H

#include <QWidget>

#include <QPushButton>

#include <QVBoxLayout>

#include <QHBoxLayout>

#include <QLineEdit>

#include <QList>

class Calculator : public QWidget

{

Q\_OBJECT

public:

Calculator();

~Calculator();

public slots:

// 数字按钮槽函数

void zeroButtonPress();

void oneButtonPress();

void twoButtonPress();

void threeButtonPress();

void fourButtonPress();

void fiveButtonPress();

void sixButtonPress();

void sevenButtonPress();

void eightButtonPress();

void nineButtonPress();

// 运算符按钮槽函数

void addButtonPress();

void subButtonPress();

void mulButtonPress();

void divButtonPress();

void clearButtonPress();

void equButtonPress();

private:

// 界面组件

QLineEdit \*operateEdit; // 显示框

QPushButton \*buttons[16]; // 按钮数组：0-9 + - \* / Clear =

// 布局管理器

QHBoxLayout \*layouts[4]; // 4行水平布局

QVBoxLayout \*mainLayout; // 主垂直布局

// 计算相关

QString expression; // 存储当前表达式

bool afterEqual; // 标记是否刚按过等号

// 辅助函数

void addDigit(char digit); // 添加数字到表达式

void addOperator(char op); // 添加运算符到表达式

};

#endif // CALCULATOR\_H

```

### calculator.cpp

```cpp

#include "calculator.h"

#include <QDebug>

#include <cmath>

Calculator::Calculator() : expression(""), afterEqual(false)

{

// 初始化显示框

operateEdit = new QLineEdit(this);

operateEdit->setReadOnly(true);

operateEdit->setText("0");

operateEdit->setAlignment(Qt::AlignRight);

operateEdit->setStyleSheet("font-size: 20px;");

// 按钮标签

const char\* buttonLabels[] = {

"7", "8", "9", "/",

"4", "5", "6", "\*",

"1", "2", "3", "-",

"0", "Clear", "=", "+"

};

// 创建按钮

for (int i = 0; i < 16; i++) {

buttons[i] = new QPushButton(tr(buttonLabels[i]), this);

buttons[i]->setMinimumSize(50, 50);

buttons[i]->setStyleSheet("font-size: 16px;");

}

// 创建布局

mainLayout = new QVBoxLayout(this);

mainLayout->addWidget(operateEdit);

// 创建4行水平布局

for (int i = 0; i < 4; i++) {

layouts[i] = new QHBoxLayout();

mainLayout->addLayout(layouts[i]);

}

// 将按钮添加到布局

for (int row = 0; row < 4; row++) {

for (int col = 0; col < 4; col++) {

layouts[row]->addWidget(buttons[row\*4 + col]);

}

}

// 连接信号槽

connect(buttons[0], SIGNAL(clicked()), this, SLOT(sevenButtonPress()));

connect(buttons[1], SIGNAL(clicked()), this, SLOT(eightButtonPress()));

connect(buttons[2], SIGNAL(clicked()), this, SLOT(nineButtonPress()));

connect(buttons[3], SIGNAL(clicked()), this, SLOT(divButtonPress()));

connect(buttons[4], SIGNAL(clicked()), this, SLOT(fourButtonPress()));

connect(buttons[5], SIGNAL(clicked()), this, SLOT(fiveButtonPress()));

connect(buttons[6], SIGNAL(clicked()), this, SLOT(sixButtonPress()));

connect(buttons[7], SIGNAL(clicked()), this, SLOT(mulButtonPress()));

connect(buttons[8], SIGNAL(clicked()), this, SLOT(oneButtonPress()));

connect(buttons[9], SIGNAL(clicked()), this, SLOT(twoButtonPress()));

connect(buttons[10], SIGNAL(clicked()), this, SLOT(threeButtonPress()));

connect(buttons[11], SIGNAL(clicked()), this, SLOT(subButtonPress()));

connect(buttons[12], SIGNAL(clicked()), this, SLOT(zeroButtonPress()));

connect(buttons[13], SIGNAL(clicked()), this, SLOT(clearButtonPress()));

connect(buttons[14], SIGNAL(clicked()), this, SLOT(equButtonPress()));

connect(buttons[15], SIGNAL(clicked()), this, SLOT(addButtonPress()));

this->setWindowTitle(tr("Calculator"));

}

Calculator::~Calculator()

{

// 清理内存

delete operateEdit;

for (int i = 0; i < 16; i++) {

delete buttons[i];

}

for (int i = 0; i < 4; i++) {

delete layouts[i];

}

delete mainLayout;

}

// 添加数字到表达式

void Calculator::addDigit(char digit)

{

if (afterEqual) {

expression.clear(); // 如果刚按过等号，则开始新表达式

afterEqual = false;

}

// 避免以0开头的多位数

if (expression.isEmpty() || expression.endsWith(' ')) {

expression.append(digit);

} else {

// 获取当前数字部分

int lastSpace = expression.lastIndexOf(' ') + 1;

QString lastNum = expression.mid(lastSpace);

// 避免多个0开头

if (!(lastNum == "0" && digit == '0')) {

if (lastNum == "0") {

expression.chop(1); // 移除多余的0

}

expression.append(digit);

}

}

operateEdit->setText(expression);

}

// 添加运算符到表达式

void Calculator::addOperator(char op)

{

if (expression.isEmpty()) {

if (op == '-') {

expression.append('-'); // 允许负号开头

operateEdit->setText(expression);

}

return;

}

// 移除可能存在的空格

expression = expression.trimmed();

// 获取最后一个字符

QChar lastChar = expression[expression.length() - 1];

// 如果最后是运算符，则替换它

if (lastChar == '+' || lastChar == '-' || lastChar == '\*' || lastChar == '/') {

expression.chop(1);

}

// 添加新运算符和空格

expression.append(' ');

expression.append(op);

expression.append(' ');

afterEqual = false;

operateEdit->setText(expression);

}

// 数字按钮处理函数

void Calculator::zeroButtonPress() { addDigit('0'); }

void Calculator::oneButtonPress() { addDigit('1'); }

void Calculator::twoButtonPress() { addDigit('2'); }

void Calculator::threeButtonPress() { addDigit('3'); }

void Calculator::fourButtonPress() { addDigit('4'); }

void Calculator::fiveButtonPress() { addDigit('5'); }

void Calculator::sixButtonPress() { addDigit('6'); }

void Calculator::sevenButtonPress() { addDigit('7'); }

void Calculator::eightButtonPress() { addDigit('8'); }

void Calculator::nineButtonPress() { addDigit('9'); }

// 运算符按钮处理函数

void Calculator::addButtonPress() { addOperator('+'); }

void Calculator::subButtonPress() { addOperator('-'); }

void Calculator::mulButtonPress() { addOperator('\*'); }

void Calculator::divButtonPress() { addOperator('/'); }

// 清屏按钮

void Calculator::clearButtonPress()

{

expression.clear();

operateEdit->setText("0");

afterEqual = false;

}

// 等于按钮 - 核心计算逻辑

void Calculator::equButtonPress()

{

if (expression.isEmpty()) {

operateEdit->setText("0");

return;

}

// 解析表达式为数字和运算符列表

QList<double> numbers;

QList<char> operators;

QStringList tokens = expression.split(' ', Qt::SkipEmptyParts);

// 有效表达式检查

if (tokens.size() < 1) {

operateEdit->setText("0");

return;

}

// 第一个元素必须是数字

bool ok;

double num = tokens[0].toDouble(&ok);

if (!ok) {

operateEdit->setText("Error");

return;

}

numbers.append(num);

// 解析后续的运算符和数字

for (int i = 1; i < tokens.size(); i += 2) {

if (i + 1 >= tokens.size()) {

// 运算符后缺少数字

operateEdit->setText("Error");

return;

}

// 解析运算符

QString opStr = tokens[i];

if (opStr.length() != 1 ||

(opStr[0] != '+' && opStr[0] != '-' &&

opStr[0] != '\*' && opStr[0] != '/')) {

operateEdit->setText("Error");

return;

}

operators.append(opStr[0].toLatin1());

// 解析数字

num = tokens[i+1].toDouble(&ok);

if (!ok) {

operateEdit->setText("Error");

return;

}

numbers.append(num);

}

// 第一步：处理所有乘除运算

for (int i = 0; i < operators.size(); ) {

if (operators[i] == '\*' || operators[i] == '/') {

double left = numbers[i];

double right = numbers[i+1];

double result;

if (operators[i] == '\*') {

result = left \* right;

} else {

if (right == 0) {

operateEdit->setText("Error: Div0");

expression.clear();

return;

}

result = left / right;

}

// 更新数字列表

numbers[i] = result;

numbers.removeAt(i+1);

operators.removeAt(i);

} else {

i++; // 跳过加减法

}

}

// 第二步：处理加减运算

double result = numbers[0];

for (int i = 0; i < operators.size(); i++) {

if (operators[i] == '+') {

result += numbers[i+1];

} else {

result -= numbers[i+1];

}

}

// 显示结果

expression = QString::number(static\_cast<int>(std::round(result))); // 四舍五入为整数

operateEdit->setText(expression);

afterEqual = true;

}

```

### main.cpp

```cpp

#include <QApplication>

#include "calculator.h"

int main(int argc, char \*argv[])

{

QApplication app(argc, argv);

// 设置应用样式

app.setStyle("Fusion");

Calculator calculator;

calculator.resize(300, 400);

calculator.show();

return app.exec();

}

```

### 代码详细解读：

#### 1. 数据结构改进

- \*\*表达式存储\*\*：使用单个QString存储完整表达式（如 "3 + 4 \* 5"）

- \*\*标记系统\*\*：`afterEqual`标记用于区分等号后的新输入

- \*\*双列表计算\*\*：使用`numbers`和`operators`列表存储解析后的表达式

#### 2. 核心计算逻辑

```cpp

// 第一步：处理乘除（优先级高）

for (int i = 0; i < operators.size(); ) {

if (operators[i] == '\*' || operators[i] == '/') {

// 计算结果并更新列表

// 注意：删除元素后索引不递增

} else {

i++; // 仅跳过加减法

}

}

// 第二步：处理加减

double result = numbers[0];

for (int i = 0; i < operators.size(); i++) {

// 顺序执行加减法

}

```

#### 3. 输入处理机制

- \*\*数字输入\*\*：

```cpp

void addDigit(char digit) {

if (afterEqual) expression.clear(); // 等号后输入清空表达式

// 处理前导零问题

}

```

- \*\*运算符输入\*\*：

```cpp

void addOperator(char op) {

// 替换最后一个运算符

if (lastChar是运算符) expression.chop(1);

// 添加空格分隔符：如 "3 + 4"

}

```

#### 4. 错误处理

- \*\*表达式解析\*\*：检查token数量有效性

- \*\*除零错误\*\*：除法操作前检查除数

- \*\*数字转换\*\*：使用`toDouble(&ok)`检查转换成功

#### 5. 界面优化

- \*\*按钮布局\*\*：使用二维数组管理按钮

- \*\*样式设置\*\*：

```cpp

operateEdit->setStyleSheet("font-size: 20px;");

buttons[i]->setMinimumSize(50, 50);

```

- \*\*显示对齐\*\*：`setAlignment(Qt::AlignRight)`

#### 6. 表达式解析

```cpp

QStringList tokens = expression.split(' ', Qt::SkipEmptyParts);

// 示例: "3 + 4 \* 5" -> ["3", "+", "4", "\*", "5"]

```

### 使用说明：

1. \*\*输入表达式\*\*：按顺序输入数字和运算符

- 示例：`3 + 4 \* 5`

2. \*\*执行计算\*\*：按`=`键计算表达式

- 结果：`23` (先计算4\*5=20，再3+20)

3. \*\*连续计算\*\*：

- 等号后按运算符：以结果开始新计算

- 等号后按数字：开始全新表达式

4. \*\*错误处理\*\*：

- 无效表达式显示"Error"

- 除零错误显示"Error: Div0"

### 改进亮点：

1. \*\*优先级处理\*\*：严格实现先乘除后加减

2. \*\*多运算符支持\*\*：支持任意长度表达式

3. \*\*实时显示\*\*：完整显示当前表达式

4. \*\*健壮性\*\*：完善的错误检查和输入验证

5. \*\*界面友好\*\*：清晰的按钮布局和显示

这个实现完全满足您的需求：无小数点、支持多运算符、先乘除后加减、按等号才计算。代码结构清晰，注释详细，适合Qt初学者学习。