# 目录

[Qt计算器v1.0 2](#_Toc28716)

[目的与要求 2](#_Toc5655)

[实验内容 2](#_Toc5574)

[1创建程序 2](#_Toc14678)

[2实现功能 5](#_Toc16895)

[3运行程序 10](#_Toc31849)

[Qt计算器v2.0 10](#_Toc1195)

[目的要求： 10](#_Toc26454)

[实验内容 10](#_Toc10033)

[Qt计算器v3.0 13](#_Toc24953)

[1目的要求 13](#_Toc28875)

[2实验内容 14](#_Toc17734)

[Qt计算器v4.0 24](#_Toc18868)

[目的要求 24](#_Toc23359)

[实验内容 24](#_Toc7375)

[实验步骤 25](#_Toc15739)

[Qt计算器v5.0 26](#_Toc5011)

[目的要求 26](#_Toc28777)

[实验内容 26](#_Toc26169)

[Qt计算器v6.0 31](#_Toc8893)

[目的要求 31](#_Toc2693)

[实验步骤 31](#_Toc21923)

# 

# Qt计算器v1.0

## 目的与要求

编写程序，实现一个简单的可以进行两个数的加减乘除的计算器

## 实验内容

编写程序，实现一个简单的可以进行两个数的加减乘除的计算器

### 1创建程序

1. 新建Qt Widgets应用，项目名称为calculator

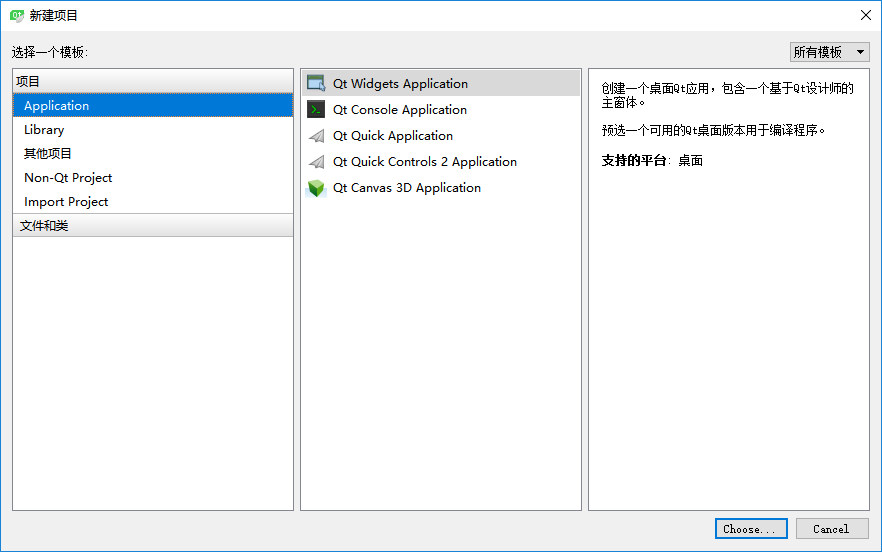


图1.1

选择QWidget作为为基类



图1.2

（2）完成项目创建后，双击widget.ui进入设计模式

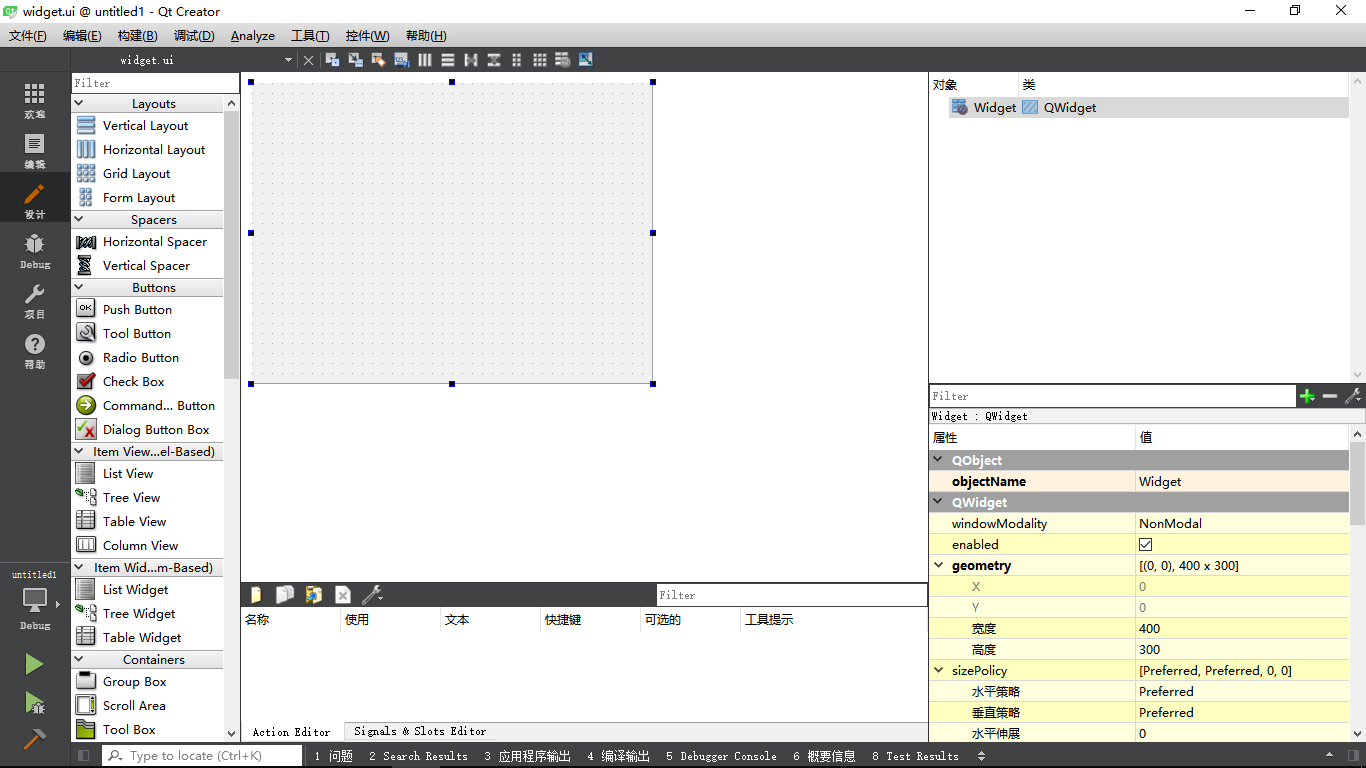


图1.3

然后向界面上拖动19个QPushButton和一个QlineEdit部件，修改其显示文本，如下图所示

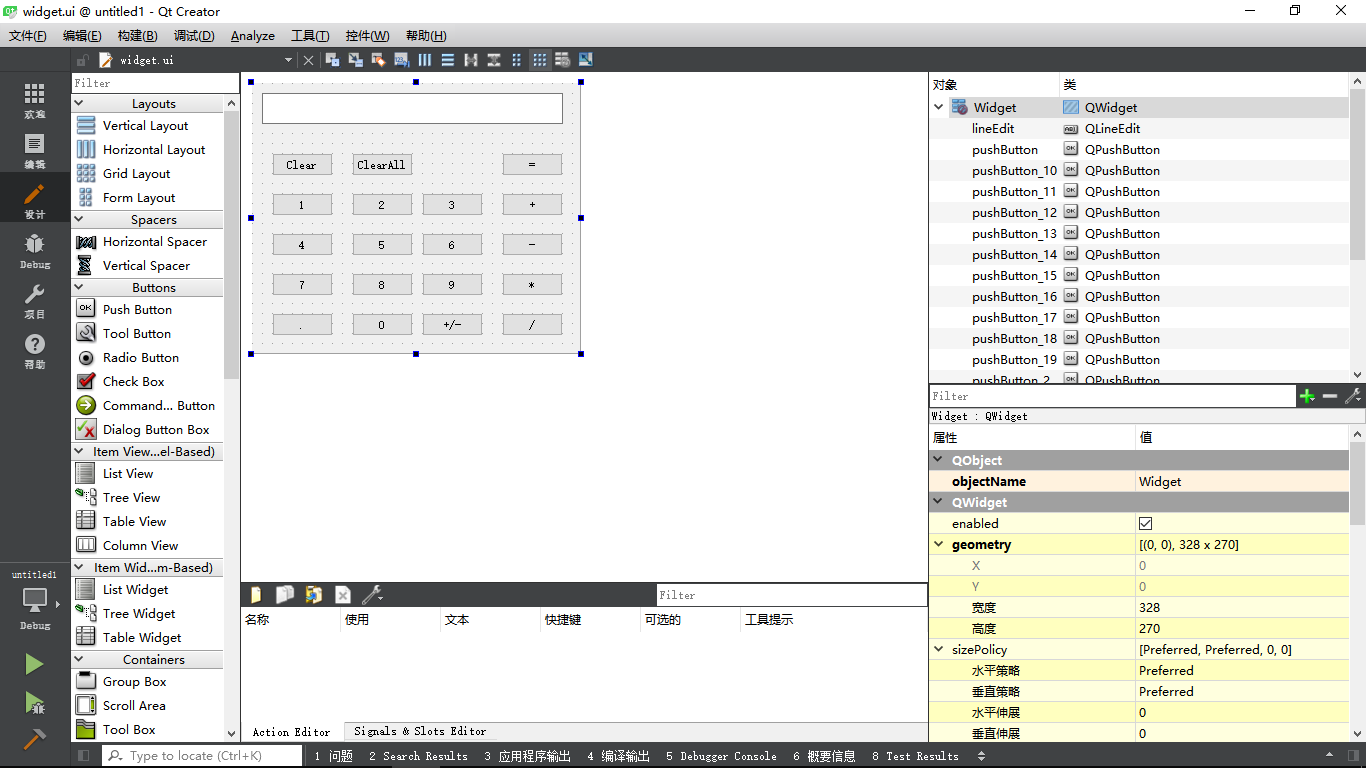


图1.4

然后按ctrl+G使用栅格布局对窗口进行布局管理

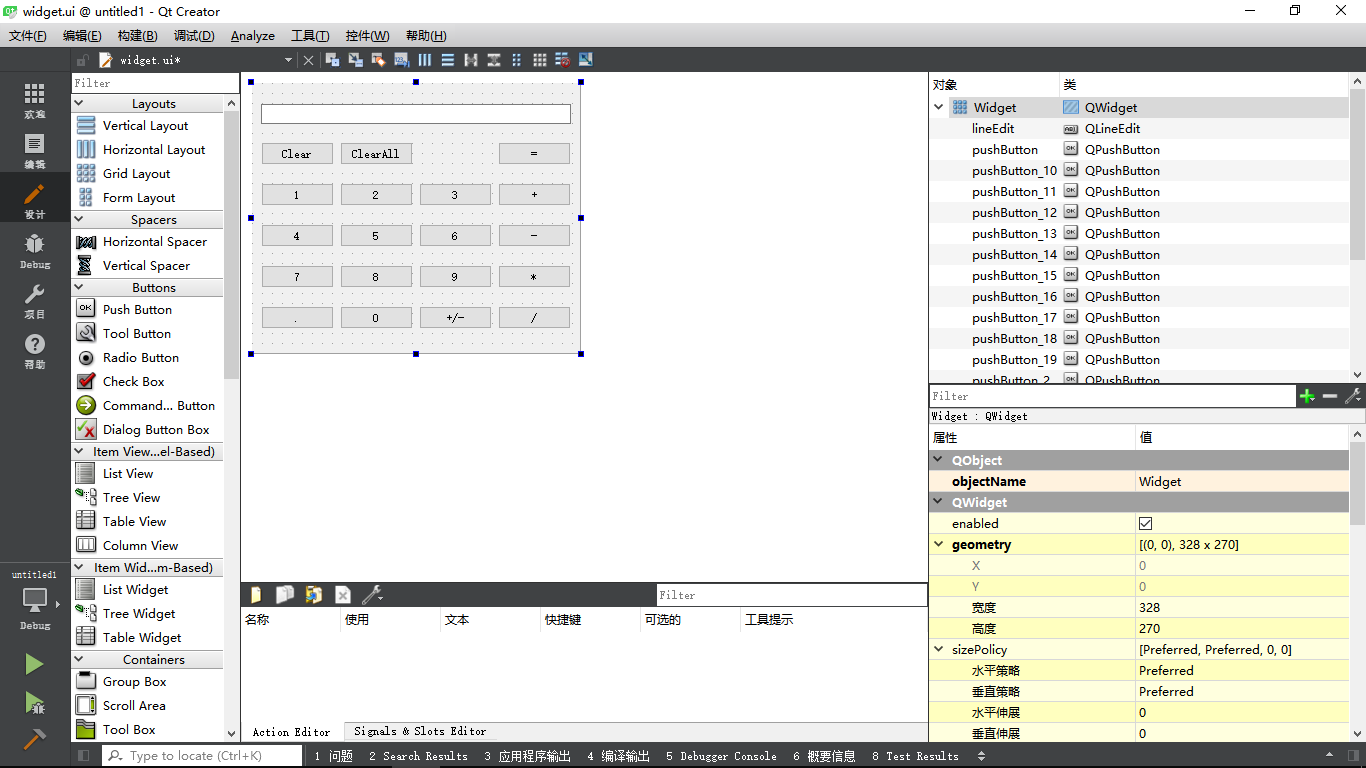


图1.5

1. 下面修改部分部件的objectName属性，将数字按钮的objectName设置为digitBtn加上表示的数字，如：digitBtn1, digitBtn2。将表示运算符号的按钮的objectName，按照加减乘除分别命名为:addBtn, subtractBtn, mulBtn, divisionBtn。将Clear和ClearAll按钮objectName设置为clearBtn和clearAllBtn。将=按钮设置为equalBtn。将小数点按钮设置为pointBtn，符号按钮设置为signBtn;
2. 选中lineEdit, 将alignment的水平的属性设置为AlignRight, 将readOnly选中，如下图1.6所示

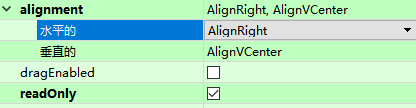


图1.6

1. 选中主窗口，将windowTitle改为Calculator,然后运行程序，确保程序如图1.7所示

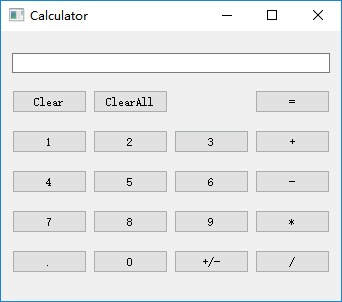


图1.7

### 2实现功能

1. 在widget.h里面添加几个私有成员和私有函数

private:

bool calculate(double operand, QString pendingOperator);

//终止运算，清除数据，报错

void abortOperation();

//连接信号和槽

void connectSlots();

//储存运算符

QString pendingOperator;

//储存运算结果

double result;

//标记是否等待一个操作数

bool waitForOperand;

在widget.cpp里面添加函数的实现

先在widget.cpp的开头添加头文件#include <QMessageBox>

bool Widget::calculate(double operand, QString pendingOperator)

{

if(pendingOperator == "+")

{

result += operand;

}

else if(pendingOperator == "-")

{

result -= operand;

}

else if(pendingOperator == "\*")

{

result \*= operand;

}

else if(pendingOperator == "/")

{

if(operand == 0.0)

return false;

result /= operand;

}

return true;

}

void Widget::abortOperation()

{

result = 0.0;

pendingOperator.clear();

ui->lineEdit->setText("0");

waitForOperand = true;

QMessageBox::warning(this, "运算错误", "除数不能为零");

}

void Widget::connectSlots()

{

QPushButton \*digitBtns[10] = {

ui->digitBtn0, ui->digitBtn1, ui->digitBtn2, ui->digitBtn3,

ui->digitBtn4, ui->digitBtn5, ui->digitBtn6, ui->digitBtn7,

ui->digitBtn8, ui->digitBtn9

};

for (auto btn : digitBtns)

connect(btn, &QPushButton::clicked, this, &Widget::digitClicked);

QPushButton \*operatorBtns[4] = {

ui->addBtn, ui->subtractionBtn, ui->mulBtn, ui->divisionBtn,

};

for (auto btn : operatorBtns)

connect(btn, &QPushButton::clicked, this, &Widget::operatorClicked);

}

（2）在widget.h里面添加几个槽函数

private slots:

void on\_clearBtn\_clicked();

void on\_clearAllBtn\_clicked();

void on\_equalBtn\_clicked();

void digitClicked();

void on\_signBtn\_clicked();

void operatorClicked();

void on\_pointBtn\_clicked();

在widget.cpp里添加槽函数实现

void Widget::digitClicked()

{

QPushButton \*digitBtn = static\_cast<QPushButton\*>(sender());

QString value = digitBtn->text();

if(ui->lineEdit->text() == "0" && value == "0")

return;

if(waitForOperand)

{

ui->lineEdit->setText(value);

waitForOperand = false;

}

else

{

ui->lineEdit->setText(ui->lineEdit->text() + value);

}

}

void Widget::on\_clearBtn\_clicked()

{

//将当前显示的数归零

ui->lineEdit->setText("0");

waitForOperand = true;

}

void Widget::on\_clearAllBtn\_clicked()

{

//将当前显示的数据归零，并将之前保存的数据运算清除

ui->lineEdit->setText("0");

waitForOperand = true;

result = 0.0;

pendingOperator.clear();

}

void Widget::on\_equalBtn\_clicked()

{

double operand = ui->lineEdit->text().toDouble();

if(pendingOperator.isEmpty())

return;

if(!calculate(operand, pendingOperator))

{

abortOperation();

return;

}

ui->lineEdit->setText(QString::number(result));

pendingOperator.clear();

result = 0.0;

waitForOperand = true;

}

void Widget::on\_signBtn\_clicked()

{

QString text = ui->lineEdit->text();

double value = text.toDouble();

if(value > 0)

{

text.prepend('-');

}

else if(value < 0)

{

text.remove(0, 1);

}

ui->lineEdit->setText(text);

}

void Widget::operatorClicked()

{

QPushButton \*clickedBtn = qobject\_cast<QPushButton \*>(sender());

QString clickedOperator = clickedBtn->text();

double operand = ui->lineEdit->text().toDouble();

if(!pendingOperator.isEmpty())

{

if(!calculate(operand, pendingOperator))

{

abortOperation();

return;

}

ui->lineEdit->setText(QString::number(result));

}

else

{

result = operand;

}

pendingOperator = clickedOperator;

waitForOperand = true;

}

void Widget::on\_pointBtn\_clicked()

{

if (waitForOperand)

ui->lineEdit->setText("0");

if (!ui->lineEdit->text().contains('.'))

ui->lineEdit->setText(ui->lineEdit->text() + ".");

waitForOperand = false;

}

1. 链接信号和槽

在Widget类的构造函数里添加代码如下

Widget::Widget(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Widget)

{

ui->setupUi(this);

ui->lineEdit->setText("0");

result = 0.0;

waitForOperand = true;

connectSlots();

}

### 3运行程序

编译运行程序，效果如图1.8所示

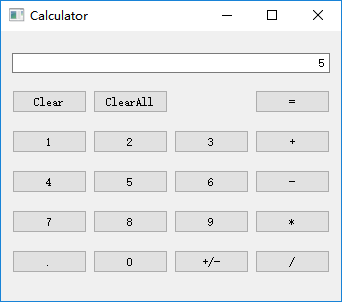


图1.8

# Qt计算器v2.0

## 目的要求：

对实验一的计算器进行改良，使其能够接受键盘输入；

## 实验内容

1. 打开实验一的项目，在widget.h私有成员里面添加一个函数

void setShortcutKeys();

（2）在widget.cpp

然后再添加代码的实现

void Widget::setShortcutKeys()

{

Qt::Key key[18] = {

Qt::Key\_0, Qt::Key\_1, Qt::Key\_2, Qt::Key\_3,

Qt::Key\_4, Qt::Key\_5, Qt::Key\_6, Qt::Key\_7,

Qt::Key\_8, Qt::Key\_9,

Qt::Key\_Plus, Qt::Key\_Minus, Qt::Key\_Asterisk, Qt::Key\_Slash,

Qt::Key\_Enter, Qt::Key\_Period, Qt::Key\_Backspace, Qt::Key\_M

};

QPushButton \*btn[18] = {

ui->digitBtn0, ui->digitBtn1, ui->digitBtn2, ui->digitBtn3,

ui->digitBtn4, ui->digitBtn5, ui->digitBtn6, ui->digitBtn7,

ui->digitBtn8, ui->digitBtn9,

ui->addBtn, ui->subtractionBtn, ui->multiplicationBtn, ui->divisionBtn,

ui->equalBtn, ui->pointBtn, ui->clearBtn, ui->signBtn

};

for (int i = 0; i < 18; i++)

btn[i]->setShortcut(QKeySequence(key[i]));

ui->clearAllBtn->setShortcut(QKeySequence("Ctrl+Backspace"));

}

在构造函数中调用改函数

Widget::Widget(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Widget)

{

initUi();

result = 0.0;

waitForOperand = true;

connectSlots();

setShortcutKeys();

}

（3）

运行程序，检查是否可以使用键盘输入数据。

# Qt计算器v3.0

## 1目的要求

1. 在Qt计算器v2.0的基础上进行完善，使得计算器能够计算一个表达式的值

## 2实验内容

1. 双击widget.ui文件，转到设计模式，先打破布局，再向界面中拖入三个新的按钮，修改内容，再重新布局，使其如图3.1所示

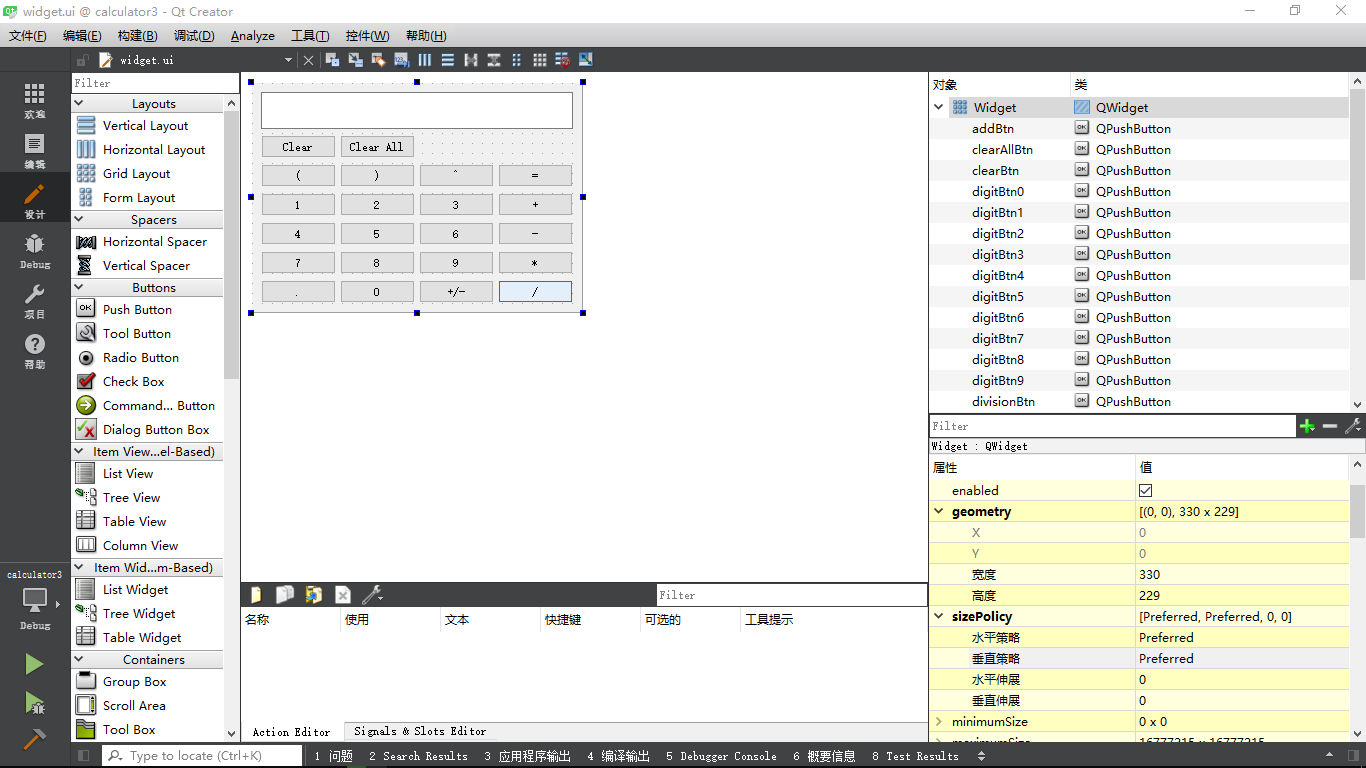


图3.1

之后修改按钮的名字，按照(, ), ^分别命名为leftBracketBtn, rightBracketBtn, powBtn.

（2）双击widget.h文件，将文件中的私有成员改为

private:

//将表达式转化为后缀表达式

QString inToPost(QString infix) throw(const char\*);

//计算后缀表达式的结果

double compute(QString s) throw(const char\*);

void abortOperation();

Ui::Widget \*ui;

bool waitForOperand;

QString error;

1. 转到widget.cpp文件中，添加几个要用到的头文件

#include <map>

#include <stack>

#include <cmath>

修改Widget构造函数的内容

改为

Widget::Widget(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Widget)

{

initUi();

waitForOperand = true;

connectSlots();

setShortcutKeys();

}

删除掉calculate函数的定义

将abortOperation的函数定义修改为

void Widget::abortOperation()

{

ui->lineEdit->setText("0");

waitForOperand = true;

QMessageBox::warning(this, "运算错误", error);

}

将digitClicked的函数修改为

void Widget::digitClicked()

{

QPushButton \*digitBtn = static\_cast<QPushButton\*>(sender());

QString value = digitBtn->text();

if(ui->lineEdit->text() == "0" && value == "0")

return;

if(waitForOperand)

{

ui->lineEdit->setText(value);

waitForOperand = false;

}

else

{

ui->lineEdit->setText(ui->lineEdit->text() + value);

}

}

将on\_equalBtn\_clicked的函数修改为

void Widget::on\_equalBtn\_clicked()

{

double result = 0.0;

try

{

result = compute(inToPost(ui->lineEdit->text()));

}

catch(const char \*er)

{

error = er;

abortOperation();

return;

}

ui->lineEdit->setText(ui->lineEdit->text() + '=' + QString::number(result));

waitForOperand = true;

}

将on\_sign\_clicked的实现修改为

void Widget::on\_signBtn\_clicked()

{

QString text = ui->lineEdit->text();

QChar sign = text[text.size() - 1];

if(sign == '-')

{

text.remove(text.size() - 1, 1);

}

else

{

text.append('-');

}

ui->lineEdit->setText(text);

}

将operatorClicked的函数实现修改为

void Widget::operatorClicked()

{

QPushButton \*clickedBtn = qobject\_cast<QPushButton \*>(sender());

QString clickedOperator = clickedBtn->text();

ui->lineEdit->setText(ui->lineEdit->text() + clickedOperator);

}

将pointClicked的函数实现修改为

void Widget::on\_pointBtn\_clicked()

{

if (waitForOperand)

ui->lineEdit->setText("0");

ui->lineEdit->setText(ui->lineEdit->text() + ".");

waitForOperand = false;

}

将setShortCutKeys的函数实现更改为

void Widget::setShortcutKeys()

{

Qt::Key key[21] = {

Qt::Key\_0, Qt::Key\_1, Qt::Key\_2, Qt::Key\_3,

Qt::Key\_4, Qt::Key\_5, Qt::Key\_6, Qt::Key\_7,

Qt::Key\_8, Qt::Key\_9,

Qt::Key\_Plus, Qt::Key\_Minus, Qt::Key\_Asterisk, Qt::Key\_Slash,

Qt::Key\_Enter, Qt::Key\_Period, Qt::Key\_Backspace, Qt::Key\_M,

Qt::Key\_ParenLeft, Qt::Key\_ParenRight, Qt::Key\_AsciiCircum

};

QPushButton \*btn[21] = {

ui->digitBtn0, ui->digitBtn1, ui->digitBtn2, ui->digitBtn3,

ui->digitBtn4, ui->digitBtn5, ui->digitBtn6, ui->digitBtn7,

ui->digitBtn8, ui->digitBtn9,

ui->addBtn, ui->subtractionBtn, ui->multiplicationBtn, ui->divisionBtn,

ui->equalBtn, ui->pointBtn, ui->clearBtn, ui->signBtn,

ui->leftBracketBtn, ui->rightBracketBtn, ui->powBtn

};

for (int i = 0; i < 21; i++)

btn[i]->setShortcut(QKeySequence(key[i]));

ui->clearAllBtn->setShortcut(QKeySequence("Ctrl+Backspace"));

}

修改connectSlots函数的实现

QPushButton \*digitBtns[10] = {

ui->digitBtn0, ui->digitBtn1, ui->digitBtn2, ui->digitBtn3,

ui->digitBtn4, ui->digitBtn5, ui->digitBtn6, ui->digitBtn7,

ui->digitBtn8, ui->digitBtn9

};

for (auto btn : digitBtns)

connect(btn, &QPushButton::clicked, this, &Widget::digitClicked);

QPushButton \*operatorBtns[7] = {

ui->addBtn, ui->subtractionBtn, ui->multiplicationBtn, ui->divisionBtn,

ui->leftBracketBtn, ui->rightBracketBtn, ui->powBtn

};

for (auto btn : operatorBtns)

connect(btn, &QPushButton::clicked, this, &Widget::operatorClicked);

最后再添加刚才在私有成员里声明的的函数实现

QString Widget::inToPost(QString infix) throw(const char\*)

{

std::stack<char> stack;

char current = 0;//读入的字符

QString postfix;//写入后缀表达式的字符串

std::map<char,int> priority;//运算符号优先级表

priority['+'] = 0;

priority['-'] = 0;

priority['\*'] = 1;

priority['/'] = 1;

priority['^'] = 2;

for(int i = 0; i < infix.length() ;++i)//逐个读取中缀表达式字符串中的字符

{

current =infix[i].toLatin1();

if(isdigit(current))//如果是数字直接输出

{

postfix.push\_back(current);

continue;

}

switch(current)

{

case '+':

case '-':

case '\*':

case '/':

case '^':

if(infix[i-1] != ')') //如果运算符的前一项不是右括号则说明前一个数字输入完毕，用#标识前面几个字符组成一个数字

{

if(infix[i-1].isDigit())

postfix.push\_back('#');

else

throw "expression is illegality";

}

if(!stack.empty())//比较目前符号与栈顶符号优先级，低于则出栈，并输出字符串

{

char tempTop=stack.top();

while(tempTop != '(' && priority[current]<=priority[tempTop])

{

stack.pop();

postfix.push\_back(tempTop);

if(stack.empty())

break;

tempTop=stack.top();

}

}

stack.push(current);//符号全部出栈或者遇到了'('或者大于栈顶符号的优先级，将新符号压入栈中

break;

case '.':

postfix.push\_back(current);

break;

case '%':

postfix.push\_back(current);

break;

case '(':

stack.push(current);//左括号直接入栈

break;

case ')':

postfix.push\_back('#');//右括号说明前方数字输入完成，标识一下

char tempTop;

tempTop=stack.top();

while(tempTop !='(')//直到栈顶元素是左括号才停止循环

{

stack.pop();

postfix.push\_back(tempTop);

tempTop=stack.top();

}

stack.pop();

break;

default:

throw "expression has illegality character";

break;

}

}

if(infix[infix.size()-1] != ')')

{

if(infix[infix.size()-1].isDigit())

postfix.push\_back('#');

else

throw "expression is illegality";

}

while(!stack.empty())

{

char tempOut=stack.top();

stack.pop();

postfix.push\_back(tempOut);

}

return postfix;

}

double Widget::compute(QString s) throw(const char\*)

{

std::stack<double> stack;

QString str;

double curr;

double temNum1;

double temNum2;

for(auto i = s.begin();i != s.end(); i++)

{

if((\*i).isDigit())

{

str.push\_back(\*i);

continue;

}

switch((\*i).toLatin1())

{

case '+':

temNum1=stack.top();

stack.pop();

temNum2=stack.top();

stack.pop();

stack.push(temNum2 + temNum1);

break;

case '-':

temNum1=stack.top();

stack.pop();

temNum2=stack.top();

stack.pop();

stack.push(temNum2 - temNum1);

break;

case '\*':

temNum1=stack.top();

stack.pop();

temNum2=stack.top();

stack.pop();

stack.push(temNum2 \* temNum1);

break;

case '/':

temNum1=stack.top();

stack.pop();

temNum2=stack.top();

stack.pop();

stack.push(temNum2 / temNum1);

break;

case '^':

temNum1=stack.top();

stack.pop();

temNum2=stack.top();

stack.pop();

stack.push(pow(temNum2, temNum1));

break;

case '.':

str.push\_back(\*i);

break;

case '#':

curr =str.toDouble();//字符串转换为浮点型

str.clear();

stack.push(curr);

break;

case '%':

curr = stack.top();

stack.pop();

curr\*=0.01;

stack.push(curr);

break;

default:

throw "expression has illegality character";

break;

}

}

curr=stack.top();

return curr;

}

1. 运行程序，如图3.2所示

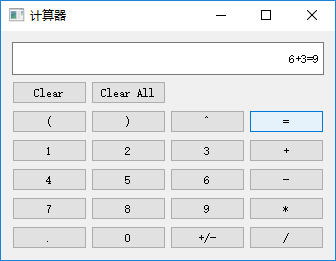


图3.2

# Qt计算器v4.0

## 目的要求

在实验三Qt计算器v3.0的基础上进行升级

可以将剪切板里的内容，复制到显示框内

## 实验内容

从本实验开始，需要在程序内添加一些特别的功能，此时，再用QWidget作为顶级窗口有些不合适了，应当选用QMainWIndow作为顶级窗口，因为QMainWindow与QWidget相比多了一些部件如菜单栏，可以用来放各种功能。

### 实验步骤

1. 新建一个项目文件，选择QMainWindow作为基类，之后，按照实验三的界面设置界面，再将实验三中的代码复制过来，但注意要改函数前面的类名要改为MainWidow.之后运行程序，确保如图3.1所示

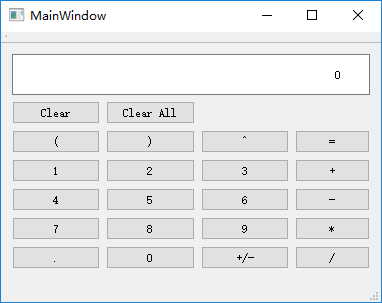


图3.1

1. 关闭程序，双击mainwindow.ui文件，点击菜单栏，在上面输入菜单，然后在菜单中添加粘贴，如图3.2所示（若无法输入中文，在别的地方写完，复制粘贴过来即可）

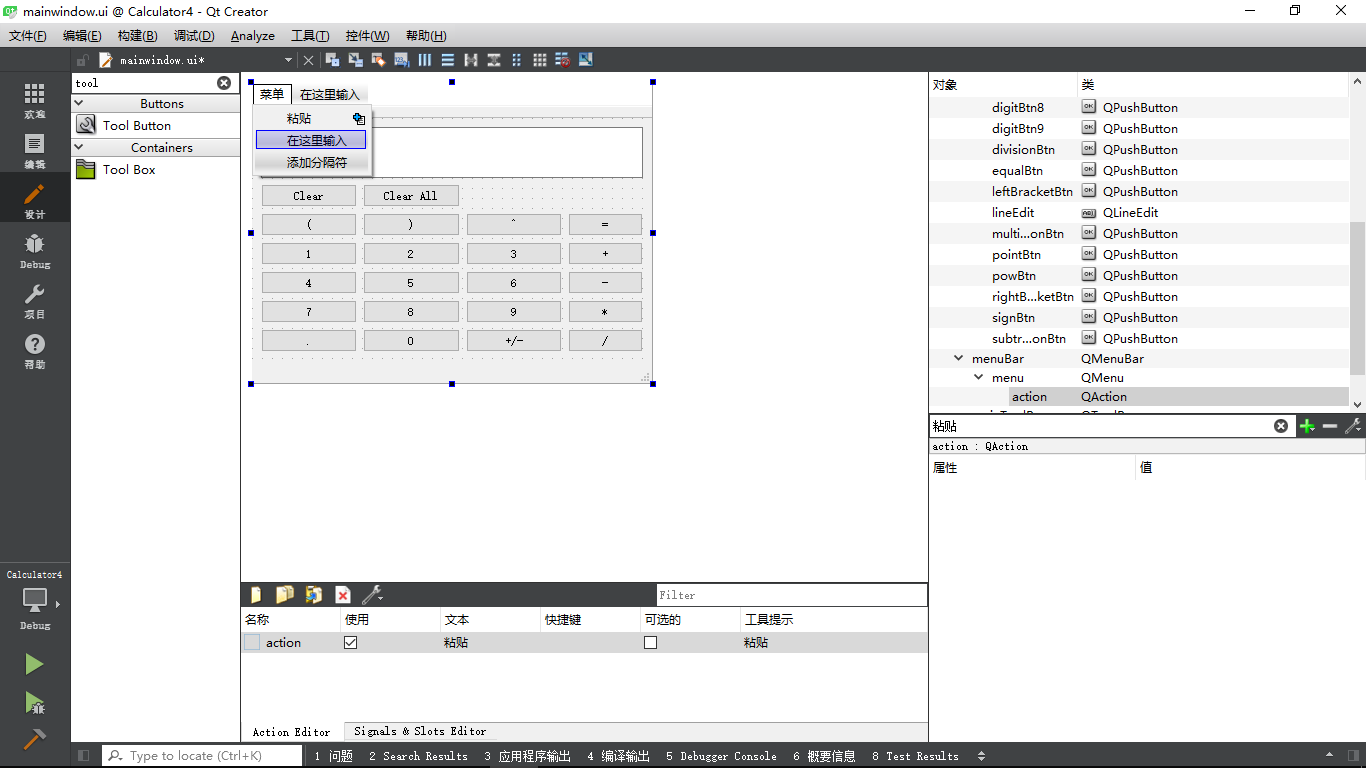
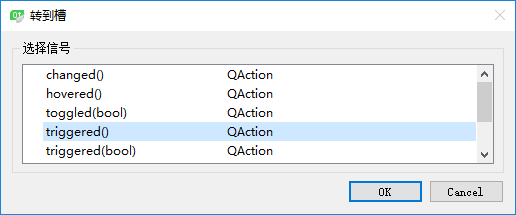


图3.2

右键点击下方的action,选择转到槽，选择triggered函数，点确定



在槽函数中添加内容

void MainWindow::on\_action\_triggered()

{

QClipboard \*board = QApplication::clipboard();

QString text = board->text();

ui->lineEdit->setText(text);

}

1. 运行程序，检查是否能够将复制的内容粘贴进输入框内。

# Qt计算器v5.0

## 目的要求

在计实验4Qt计算器v4.0的基础上进行改进，使得计算器能够记录计算历史

并且能够随时调出和消除历史

## 实验内容

1. 打开实验四的项目文件，双击mainwindow.ui, 将QLineEdit输入框删除，拖入一个QTextEdit作为输入框。之后在转到，mainwindow.h中，添加两个私有函数

//获取当前表达式的值

QString currentText();

//设置当前行的值

void setCurrentText(QString text);

和一个私有成员变量

QString log;

将bool waitForOperand改为 bool waitForOperator

这是因为，从此版本开始，每次将计算结果打印到下一行，使得可以继续进行计算，这时需要判断是否程序正在等待用户输入一个运算符。

之后转到mainwindow.cpp中添加头文件

#include <QTextBlock>

修改构造函数为

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

setCurrentText("0");

waitForOperator = false;

connectSlots();

setShortcutKeys();

}

需要修改的函数如下

void MainWindow::abortOperation()

{

setCurrentText("0");

waitForOperator = true;

statusBar()->showMessage(error, 4000);

}

void MainWindow::digitClicked()

{

QPushButton \*digitBtn = static\_cast<QPushButton\*>(sender());

QString value = digitBtn->text();

QString current = currentText();

if(current == "0" && value == "0")

return;

if(waitForOperator)

{

log.append(current + '\n');

ui->textEdit->append(value);

waitForOperator = false;

}

else

{

if(current != "0")

setCurrentText(current + value);

else

setCurrentText(value);

}

}

void MainWindow::on\_clearBtn\_clicked()

{

//将当前显示的数归零

setCurrentText("0");

waitForOperator = true;

}

void MainWindow::on\_clearAllBtn\_clicked()

{

//将当前显示的数据归零，并将之前保存的数据运算清除

ui->textEdit->setText("0");

waitForOperator = true;

}

void MainWindow::on\_equalBtn\_clicked()

{

double result = 0.0;

try

{

result = compute(inToPost(currentText()));

}

catch(const char \*er)

{

error = er;

abortOperation();

return;

}

log.append(currentText() + '\n');

ui->textEdit->append(QString::number(result));

waitForOperator = true;

}

void MainWindow::on\_signBtn\_clicked()

{

QString text = currentText();

QChar sign = text[text.size() - 1];

if(sign == '-')

{

text.remove(text.size() - 1, 1);

}

else

{

text.append('-');

}

setCurrentText(text);

}

void MainWindow::operatorClicked()

{

QPushButton \*clickedBtn = qobject\_cast<QPushButton \*>(sender());

QString clickedOperator = clickedBtn->text();

setCurrentText(currentText() + clickedOperator);

waitForOperator = false;

}

void MainWindow::on\_pointBtn\_clicked()

{

if (waitForOperator)

setCurrentText("0");

setCurrentText(currentText() + ".");

waitForOperator = false;

}

再添加两个函数实现

QString MainWindow::currentText()

{

QTextDocument \*document = ui->textEdit->document();

QTextBlock block = document->lastBlock();

return block.text();

}

void MainWindow::setCurrentText(QString text)

{

QString allText = ui->textEdit->toPlainText();

int pos = allText.lastIndexOf('\n');

if(pos >= 0)

allText.replace(allText.lastIndexOf('\n') + 1, allText.size(), text);

else

allText = text;

ui->textEdit->setText(allText);

}

之后编译，确保程序能够运行

1. 现在程序能够记录历史了，但清除后历史就消失了，现在添加一个功能，让程序可以随时将记录的历史显示出来，而且还能将记录的历史清除。双击mainwindow.ui,像之前一样，向菜单中添加两个“历史记录”、“清除记录”。然后再拖动下方的action，将“历史纪录”和“清除记录”拖动到工具栏中。如图5.1所示。

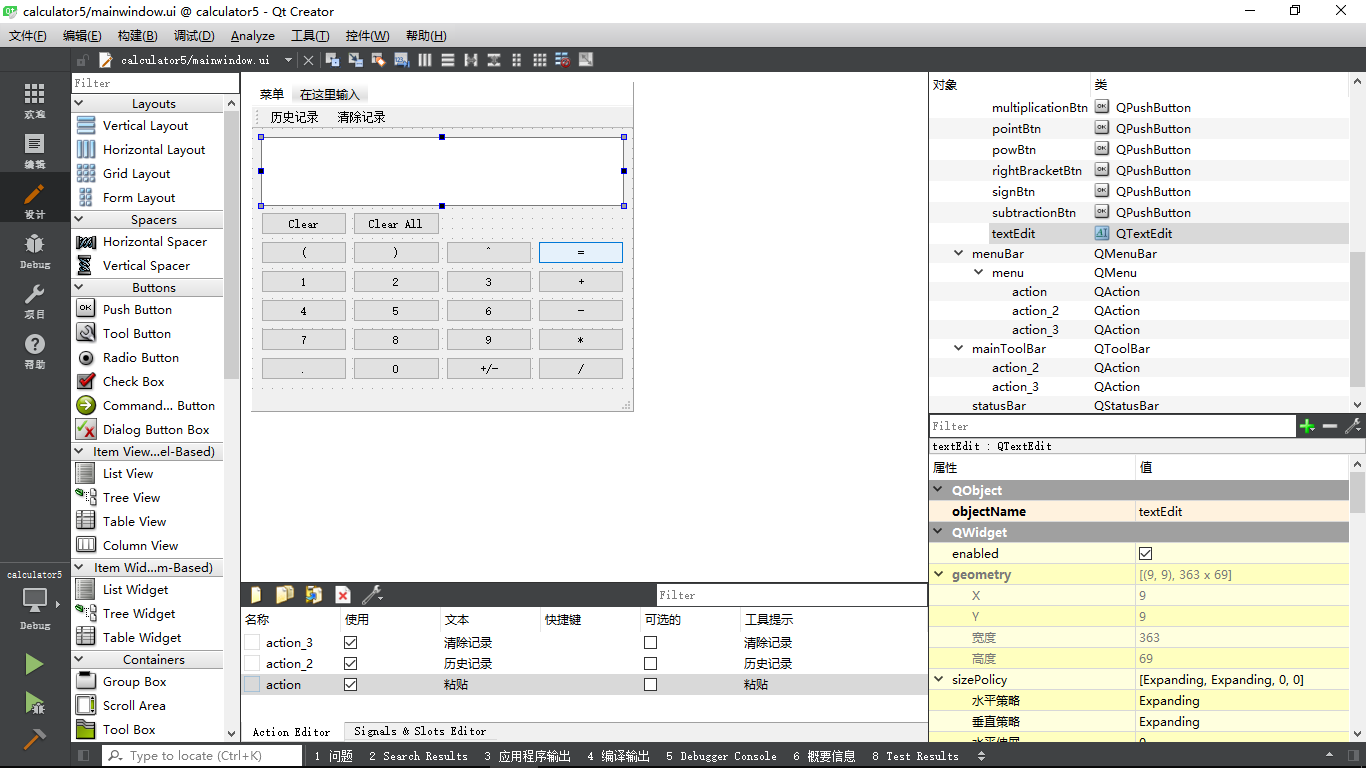


图5.1

右键点击下方的历史记录和清除记录，选择转到槽，选择triggered信号，分别添加槽函数的实现

void MainWindow::on\_action\_2\_triggered()

{

ui->textEdit->setText(log);

setCurrentText("0");

waitForOperator = false;

}

void MainWindow::on\_action\_3\_triggered()

{

log.clear();

}

（3）运行程序，查看功能是否实现。

# Qt计算器v6.0

## 目的要求

在实验5Qt计算器v5.0的基础上进行改进，使得计算拥有更多的功能，并且在用户不需要的情况下隐藏这些功能。

## 实验步骤

（1）

打开实验5的项目文件，双击mainwindow.ui进入设计模式，打破原有布局，向界面中再拖入6个按钮，并修改其显示文本为%，sin，cos，tan，ln，lg。再向“菜单”中添加一个Action“科学计算器”，并将其属性中的checkable选中。之后将所有按钮选中，应用栅格布局，再选中主窗口应用纵向布局。确保如图6.1所示

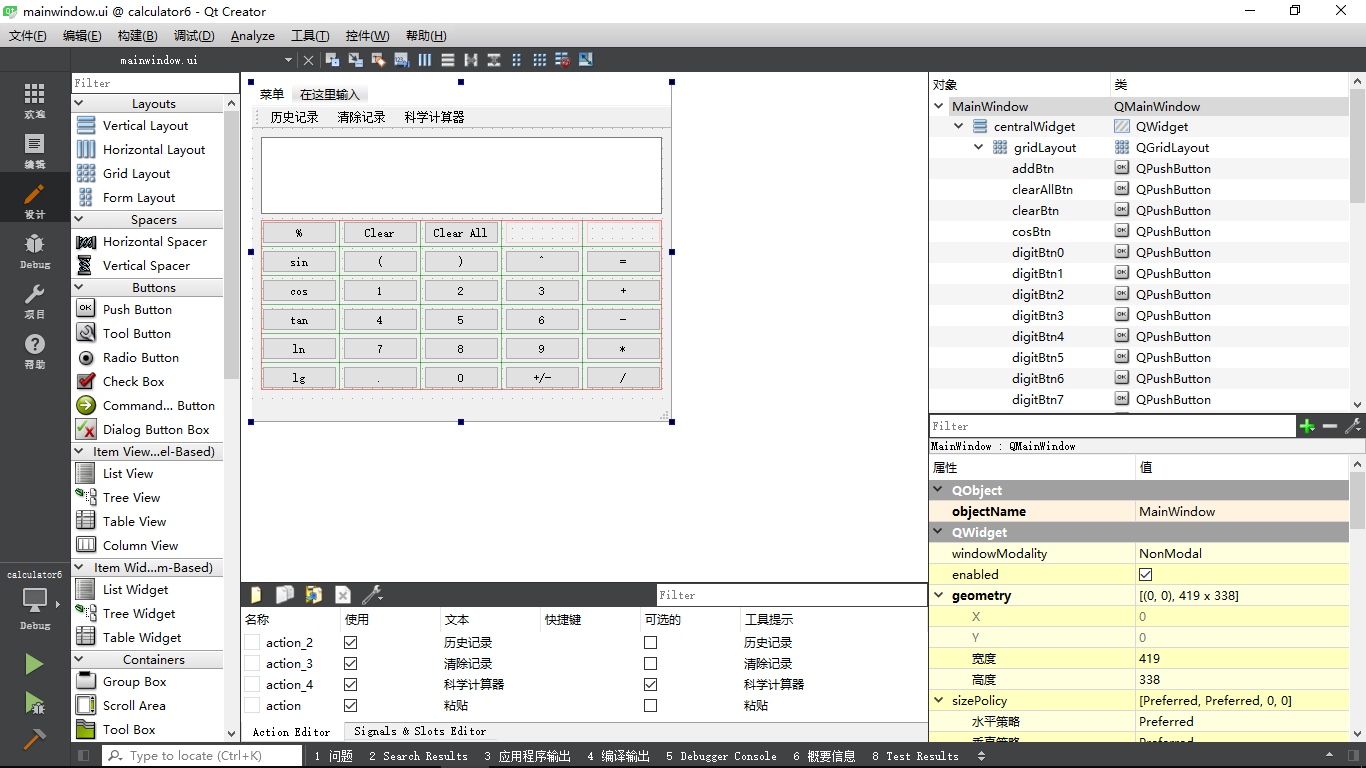


图6.1

之后按照%，sin，cos，tan，ln，lg将按钮分别命名为percentBtn, sinBtn, cosBtn, tanBtn, lnBtn, lgBtn.

选中Action“科学计算器”选择转到槽，选择triggered(bool)信号，添加槽函数实现如下

void MainWindow::on\_action\_4\_triggered(bool checked)

{

if(checked)

{

ui->percentBtn->show();

ui->sinBtn->show();

ui->cosBtn->show();

ui->tanBtn->show();

ui->lnBtn->show();

ui->lgBtn->show();

}

else

{

ui->percentBtn->hide();

ui->sinBtn->hide();

ui->cosBtn->hide();

ui->tanBtn->hide();

ui->lnBtn->hide();

ui->lgBtn->hide();

}

}

1. 转到mainwindow.h中，添加一个槽函数，

void specialOperatorClicked();

之后转到mainwindow.cpp中，添加函数实现

void MainWindow::specialOperatorClicked()

{

QPushButton \*specialOperatorBtn = static\_cast<QPushButton\*>(sender());

QString value = specialOperatorBtn->text();

QString current = currentText();

if(waitForOperator)

{

log.append(current + '\n');

ui->textEdit->append(value);

waitForOperator = false;

}

else

{

if(current != "0")

setCurrentText(current + value);

else

setCurrentText(value);

}

}

部分有改动的函数如下

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

initUi();

waitForOperator = false;

on\_action\_4\_triggered(false);

connectSlots();

setShortcutKeys();

}

void MainWindow::connectSlots()

{

QPushButton \*digitBtns[10] = {

ui->digitBtn0, ui->digitBtn1, ui->digitBtn2, ui->digitBtn3,

ui->digitBtn4, ui->digitBtn5, ui->digitBtn6, ui->digitBtn7,

ui->digitBtn8, ui->digitBtn9

};

for (auto btn : digitBtns)

connect(btn, &QPushButton::clicked, this, &MainWindow::digitClicked);

QPushButton \*operatorBtns[8] = {

ui->addBtn, ui->subtractionBtn, ui->multiplicationBtn, ui->divisionBtn,

ui->leftBracketBtn, ui->rightBracketBtn, ui->powBtn, ui->percentBtn

};

for (auto btn : operatorBtns)

connect(btn, &QPushButton::clicked, this, &MainWindow::operatorClicked);

QPushButton \*specialOperatorBtns[5] = {

ui->sinBtn, ui->cosBtn, ui->tanBtn, ui->lnBtn,

ui->lgBtn

};

for (auto btn : specialOperatorBtns)

connect(btn, &QPushButton::clicked, this, &MainWindow::specialOperatorClicked);

}

void MainWindow::setShortcutKeys()

{

Qt::Key key[27] = {

Qt::Key\_0, Qt::Key\_1, Qt::Key\_2, Qt::Key\_3,

Qt::Key\_4, Qt::Key\_5, Qt::Key\_6, Qt::Key\_7,

Qt::Key\_8, Qt::Key\_9,

Qt::Key\_Plus, Qt::Key\_Minus, Qt::Key\_Asterisk, Qt::Key\_Slash,

Qt::Key\_Enter, Qt::Key\_Period, Qt::Key\_Backspace, Qt::Key\_M,

Qt::Key\_ParenLeft, Qt::Key\_ParenRight, Qt::Key\_AsciiCircum,Qt::Key\_Percent,

Qt::Key\_S, Qt::Key\_C, Qt::Key\_T, Qt::Key\_N,

Qt::Key\_G

};

QPushButton \*btn[27] = {

ui->digitBtn0, ui->digitBtn1, ui->digitBtn2, ui->digitBtn3,

ui->digitBtn4, ui->digitBtn5, ui->digitBtn6, ui->digitBtn7,

ui->digitBtn8, ui->digitBtn9,

ui->addBtn, ui->subtractionBtn, ui->multiplicationBtn, ui->divisionBtn,

ui->equalBtn, ui->pointBtn, ui->clearBtn, ui->signBtn,

ui->leftBracketBtn, ui->rightBracketBtn, ui->powBtn, ui->percentBtn,

ui->sinBtn, ui->cosBtn, ui->tanBtn, ui->lnBtn,

ui->lgBtn

};

for (int i = 0; i < 27; i++)

btn[i]->setShortcut(QKeySequence(key[i]));

ui->clearAllBtn->setShortcut(QKeySequence("Ctrl+Backspace"));

}

QString MainWindow::inToPost(QString infix) throw(const char\*)

{

std::stack<char> stack;

char current = 0;//读入的字符

QString postfix;//写入后缀表达式的字符串

std::map<char,int> priority;//运算符号优先级表

priority['+'] = 0;

priority['-'] = 0;

priority['\*'] = 1;

priority['/'] = 1;

priority['^'] = 2;

priority['s'] = 3;

priority['c'] = 3;

priority['t'] = 3;

priority['n'] = 3;

priority['g'] = 3;

priority['%'] = 3;

for(int i = 0; i < infix.length() ;++i)//逐个读取中缀表达式字符串中的字符

{

current =infix[i].toLatin1();

if(isdigit(current))//如果是数字直接输出

{

postfix.push\_back(current);

continue;

}

switch(current)

{

case '+':

case '-':

case '\*':

case '/':

case '^':

case '%':

if(infix[i-1] != ')') //如果运算符的前一项不是右括号则说明前一个数字输入完毕，用#标识前面几个字符组成一个数字

{

if(infix[i-1].isDigit())

postfix.push\_back('#');

else if(infix[i-1] == '%')// 什么也不做，跳出

infix.begin();

else

throw "表达式是非法的";

}

if(!stack.empty())//比较目前符号与栈顶符号优先级，低于则出栈，并输出字符串

{

char tempTop=stack.top();

while(tempTop != '(' && priority[current]<priority[tempTop])

{

stack.pop();

postfix.push\_back(tempTop);

if(stack.empty())

break;

tempTop=stack.top();

}

}

stack.push(current);//符号全部出栈或者遇到了'('或者大于栈顶符号的优先级，将新符号压入栈中

break;

case 's':

case 'c':

case 't':

if(i > 0 && infix[i-1].isDigit())

throw "表达式非法";

if(!stack.empty())//比较目前符号与栈顶符号优先级，低于则出栈，并输出字符串

{

char tempTop=stack.top();

while(tempTop != '(' && priority[current]<priority[tempTop])

{

stack.pop();

postfix.push\_back(tempTop);

if(stack.empty())

break;

tempTop=stack.top();

}

}

stack.push(current);

i+=2;

break;

case 'l':

if(infix[i-1].isDigit())

throw "表达式非法";

if(!stack.empty())//比较目前符号与栈顶符号优先级，低于则出栈，并输出字符串

{

char tempTop=stack.top();

while(tempTop != '(' && priority[infix[i+1].toLatin1()] < priority[tempTop])

{

stack.pop();

postfix.push\_back(tempTop);

if(stack.empty())

break;

tempTop=stack.top();

}

}

stack.push(infix[i+1].toLatin1());

i++;

break;

case '.':

postfix.push\_back(current);

break;

case '(':

stack.push(current);//左括号直接入栈

break;

case ')':

postfix.push\_back('#');//右括号说明前方数字输入完成，标识一下

char tempTop;

tempTop=stack.top();

while(tempTop !='(')//直到栈顶元素是左括号才停止循环

{

stack.pop();

postfix.push\_back(tempTop);

tempTop=stack.top();

}

stack.pop();

break;

default:

throw "表达式中含有非法字符";

break;

}

}

if(infix[infix.size()-1] != ')')

{

if(infix[infix.size()-1].isDigit())

postfix.push\_back('#');

else if(infix[infix.size()-1] == '%') // 什么也不做

infix.begin();

else

throw "表达式非法";

}

while(!stack.empty())

{

char tempOut=stack.top();

stack.pop();

postfix.push\_back(tempOut);

}

return postfix;

}

double MainWindow::compute(QString s) throw(const char\*)

{

//qDebug() << s;

const double PI = std::acos(-1.0);

std::stack<double> stack;

QString str;

double curr;

double temNum1;

double temNum2;

for(auto i = s.begin();i != s.end(); i++)

{

if((\*i).isDigit())

{

str.push\_back(\*i);

continue;

}

switch((\*i).toLatin1())

{

case '+':

temNum1=stack.top();

stack.pop();

temNum2=stack.top();

stack.pop();

stack.push(temNum2 + temNum1);

break;

case '-':

temNum1=stack.top();

stack.pop();

temNum2=stack.top();

stack.pop();

stack.push(temNum2 - temNum1);

break;

case '\*':

temNum1=stack.top();

stack.pop();

temNum2=stack.top();

stack.pop();

stack.push(temNum2 \* temNum1);

break;

case '/':

temNum1=stack.top();

stack.pop();

temNum2=stack.top();

stack.pop();

stack.push(temNum2 / temNum1);

break;

case '^':

temNum1=stack.top();

stack.pop();

temNum2=stack.top();

stack.pop();

stack.push(pow(temNum2, temNum1));

break;

case 's':

temNum1 = stack.top();

stack.pop();

stack.push(std::sin(temNum1/180.0\*PI));

break;

case 'c':

temNum1 = stack.top();

stack.pop();

stack.push(std::cos(temNum1/180.0\*PI));

break;

case 't':

temNum1 = stack.top();

stack.pop();

stack.push(std::tan(temNum1/180.0\*PI));

break;

case 'n':

temNum1 = stack.top();

stack.pop();

stack.push(std::log(temNum1));

break;

case 'g':

temNum1 = stack.top();

stack.pop();

stack.push(std::log10(temNum1));

break;

case '.':

str.push\_back(\*i);

break;

case '#':

curr =str.toDouble();//字符串转换为浮点型

str.clear();

stack.push(curr);

break;

case '%':

curr = stack.top();

stack.pop();

curr\*=0.01;

stack.push(curr);

break;

default:

throw "表达式中含有非法字符";

break;

}

}

curr=stack.top();

return curr;

}

（3）运行程序，测试程序功能。