Ministerul Educaţiei şi Ştiinţei
al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatică şi Microelectronică
Tehnologii Informationale

Lucrare de laborator nr.2

la MIDPS

A efectuat: st.grupei TI-152
Staver Vasile

A verificat: Cojanu Irina

Chişinău 2017

Obiective:

- Realizeaza un simplu GUI Calculator
- Operatiile simple: +,-,*,/,putere,radical,InversareSemn(+/-),operatii cu numere zecimale.
- Divizare proiectului in doua module - Interfata grafica(Modul GUI) si Modulul de baza(Core Module).

Laboratory Requirements:

- *Basic Level* (nota 5 || 6):
  - Realizeaza un simplu GUI calculator care suporta functiile de baza: +, -, /, *.
- *Normal Level* (nota 7 || 8):
  - Realizeaza un simplu GUI calculator care suporta urmatoare functii: +, -, /, *, putere, radical, InversareSemn(+/-).
- *Advanced Level* (nota 9 || 10):
  - Realizeaza un simplu GUI calculator care suporta urmatoare functii: +, -, /, *, putere, radical, InversareSemn(+/-), operatii cu numere zecimale.
  - Divizare proiectului in doua module - Interfata grafica(Modul GUI) si Modulul de baza(Core Module).

Listing-ul programului :

Button.h

```
#ifndef BUTTON_H
#define BUTTON_H

#include <QToolButton>

class Button : public QToolButton
{
    Q_OBJECT

public:
    explicit Button(const QString &text, QWidget *parent = 0);

    QSize sizeHint() const override;
};

#endif
```

Calculator.h

```
#ifndef CALCULATOR_H
#define CALCULATOR_H

#include <QWidget>

class QLineEdit;

class Button;

class Calculator : public QWidget
{
    Q_OBJECT

public:
    Calculator(QWidget *parent = 0);

private slots:
    void digitClicked();
    void unaryOperatorClicked();
    void additiveOperatorClicked();
    void multiplicativeOperatorClicked();
    void equalClicked();
    void pointClicked();
    void changeSignClicked();
    void backspaceClicked();
    void clear();
    void clearAll();
    void clearMemory();
    void readMemory();
    void setMemory();
    void addToMemory();

private:
    Button *createButton(const QString &text, const char *member);
    void abortOperation();
    bool calculate(double rightOperand, const QString &pendingOperator);

    double sumInMemory;
    double sumSoFar;
    double factorSoFar;
    QString pendingAdditiveOperator;
    QString pendingMultiplicativeOperator;
    bool waitingForOperand;

    QLineEdit *display;

    enum { NumDigitButtons = 10 };
    Button *digitButtons[NumDigitButtons];
};

#endif
```

Button.cpp

```
#include <QtWidgets>

#include "button.h"

Button::Button(const QString &text, QWidget *parent)
    : QToolButton(parent)
{
    setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Preferred);
    setText(text);




}

QSize Button::sizeHint() const
{
    QSize size = QToolButton::sizeHint();
    size.rheight() += 40;
    size.rwidth() = qMax(size.width(), size.height());
    return size;
}
```

Calculator.cpp

```
#include <QtWidgets>

#include <cmath>

#include "button.h"
#include "calculator.h"

Calculator::Calculator(QWidget *parent)
    : QWidget(parent)
{
    sumInMemory = 0.0;
    sumSoFar = 0.0;
    factorSoFar = 0.0;
    waitingForOperand = true;

    display = new QLineEdit("0");
    display->setReadOnly(true);
    display->setAlignment(Qt::AlignRight);
    display->setMaxLength(10);
    display->setTextMargins(0, 30, 5, 0);
    QFont font = display->font();
    font.setPointSize(font.pointSize() + 21);
    display->setFont(font);
    QFont font1;
    font1.setPointSize(font1.pointSize() + 5);

    for (int i = 0; i < NumDigitButtons; ++i) {
        digitButtons[i] = createButton(QString::number(i), SLOT(digitClicked()));
        digitButtons[i]->setFont(font1);
    }


    Button *pointButton = createButton(".", SLOT(pointClicked()));
    pointButton->setStyleSheet("* { background-color: rgb(255,165,0) }");
    pointButton->setFont(font1);

    Button *changeSignButton = createButton("\302\261", SLOT(changeSignClicked()));
    changeSignButton->setStyleSheet("* { background-color: rgb(255,165,0) }");
    changeSignButton->setFont(font1);

    Button *backspaceButton = createButton("Backspace", SLOT(backspaceClicked()));
    backspaceButton->setStyleSheet("* { background-color: rgb(170,170,170) }");
    backspaceButton->setFont(font1);


    Button *clearAllButton = createButton(tr("Clear All"), SLOT(clearAll()));
    clearAllButton->setStyleSheet("* { background-color: rgb(170,170,170) }");
    clearAllButton->setFont(font1);


    Button *divisionButton = createButton("\303\267",
SLOT(multiplicativeOperatorClicked()));
    divisionButton->setStyleSheet("* { background-color: rgb(255,165,0) }");
     divisionButton->setFont(font1);

    Button *timesButton = createButton("\303\227",
SLOT(multiplicativeOperatorClicked()));
    timesButton->setStyleSheet("* { background-color: rgb(255,165,0) }");
     timesButton->setFont(font1);

    Button *minusButton = createButton("-", SLOT(additiveOperatorClicked()));
    minusButton->setStyleSheet("* { background-color: rgb(255,165,0) }");
     minusButton->setFont(font1);

    Button *plusButton = createButton("+", SLOT(additiveOperatorClicked()));
    plusButton->setStyleSheet("* { background-color: rgb(255,165,0) }");
     plusButton->setFont(font1);

    Button *squareRootButton = createButton("Sqrt", SLOT(unaryOperatorClicked()));
    squareRootButton->setStyleSheet("* { background-color: rgb(255,165,0) }");
     squareRootButton->setFont(font1);

    Button *powerButton = createButton("x\302\262", SLOT(unaryOperatorClicked()));
    powerButton->setStyleSheet("* { background-color: rgb(255,165,0) }");
     powerButton->setFont(font1);

    Button *reciprocalButton = createButton("1/x", SLOT(unaryOperatorClicked()));
    reciprocalButton->setStyleSheet("* { background-color: rgb(255,165,0) }");
     reciprocalButton->setFont(font1);

    Button *equalButton = createButton("=", SLOT(equalClicked()));
    equalButton->setStyleSheet("* { background-color: rgb(170,170,170) }");
     equalButton->setFont(font1);
    QGridLayout *mainLayout = new QGridLayout;
    mainLayout->setSizeConstraint(QLayout::SetFixedSize);
    mainLayout->addWidget(display, 0, 0, 1, 6);
    mainLayout->addWidget(backspaceButton, 1, 1, 1, 3);
    mainLayout->addWidget(clearAllButton, 1, 4, 1, 2);

    for (int i = 1; i < NumDigitButtons; ++i) {
        int row = ((9 - i) / 3) + 2;
        int column = ((i - 1) % 3) + 1;
        mainLayout->addWidget(digitButtons[i], row, column);
    }

    mainLayout->addWidget(digitButtons[0], 5, 1);
    mainLayout->addWidget(pointButton, 5, 2);
    mainLayout->addWidget(changeSignButton, 5, 3);

    mainLayout->addWidget(divisionButton, 2, 4);
    mainLayout->addWidget(timesButton, 3, 4);
    mainLayout->addWidget(minusButton, 4, 4);
    mainLayout->addWidget(plusButton, 5, 4);

    mainLayout->addWidget(squareRootButton, 2, 5);
    mainLayout->addWidget(powerButton, 3, 5);
    mainLayout->addWidget(reciprocalButton, 4, 5);
    mainLayout->addWidget(equalButton, 5, 5);
    setLayout(mainLayout);

    setWindowTitle("Calculator");
}

void Calculator::digitClicked()
{
    Button *clickedButton = qobject_cast<Button *>(sender());
    int digitValue = clickedButton->text().toInt();
    if (display->text() == "0" && digitValue == 0.0)
        return;

    if (waitingForOperand) {
        display->clear();
        waitingForOperand = false;
    }
    display->setText(display->text() + QString::number(digitValue));
}

void Calculator::unaryOperatorClicked()
{
    Button *clickedButton = qobject_cast<Button *>(sender());
    QString clickedOperator = clickedButton->text();
    double operand = display->text().toDouble();
    double result = 0.0;

    if (clickedOperator == tr("Sqrt")) {
        if (operand < 0.0) {
            abortOperation();
            return;
        }
        result = std::sqrt(operand);
    } else if (clickedOperator == tr("x\302\262")) {
        result = std::pow(operand, 2.0);
    } else if (clickedOperator == tr("1/x")) {
        if (operand == 0.0) {
            abortOperation();
            return;
        }
```

```
            result = 1.0 / operand;
        }
        display->setText(QString::number(result));
        waitingForOperand = true;
    }

    void Calculator::additiveOperatorClicked()
    {
        Button *clickedButton = qobject_cast<Button *>(sender());
        QString clickedOperator = clickedButton->text();
        double operand = display->text().toDouble();

        if (!pendingMultiplicativeOperator.isEmpty()) {
            if (!calculate(operand, pendingMultiplicativeOperator)) {
                abortOperation();
                return;
            }
            display->setText(QString::number(factorSoFar));
            operand = factorSoFar;
            factorSoFar = 0.0;
            pendingMultiplicativeOperator.clear();
        }

        if (!pendingAdditiveOperator.isEmpty()) {
            if (!calculate(operand, pendingAdditiveOperator)) {
                abortOperation();
                return;
            }
            display->setText(QString::number(sumSoFar));
        } else {
            sumSoFar = operand;
        }

        pendingAdditiveOperator = clickedOperator;
        waitingForOperand = true;
    }

    void Calculator::multiplicativeOperatorClicked()
    {
        Button *clickedButton = qobject_cast<Button *>(sender());
        QString clickedOperator = clickedButton->text();
        double operand = display->text().toDouble();

        if (!pendingMultiplicativeOperator.isEmpty()) {
            if (!calculate(operand, pendingMultiplicativeOperator)) {
                abortOperation();
                return;
            }
            display->setText(QString::number(factorSoFar));
        } else {
            factorSoFar = operand;
        }

        pendingMultiplicativeOperator = clickedOperator;
        waitingForOperand = true;
    }

    void Calculator::equalClicked()
    {
        double operand = display->text().toDouble();

        if (!pendingMultiplicativeOperator.isEmpty()) {
            if (!calculate(operand, pendingMultiplicativeOperator)) {
                abortOperation();
                return;
            }
            operand = factorSoFar;
            factorSoFar = 0.0;
            pendingMultiplicativeOperator.clear();
        }
        if (!pendingAdditiveOperator.isEmpty()) {
            if (!calculate(operand, pendingAdditiveOperator)) {
                abortOperation();
                return;
            }
            pendingAdditiveOperator.clear();
        } else {
            sumSoFar = operand;
        }

        display->setText(QString::number(sumSoFar, 'g',5));
        sumSoFar = 0.0;
        waitingForOperand = true;
    }

    void Calculator::pointClicked()
    {
        if (waitingForOperand)
            display->setText("0");
        if (!display->text().contains('.'))
            display->setText(display->text() + tr("."));
        waitingForOperand = false;
    }

    void Calculator::changeSignClicked()
    {
        QString text = display->text();
        double value = text.toDouble();

        if (value > 0.0) {
            text.prepend(tr("-"));
        } else if (value < 0.0) {
            text.remove(0, 1);
        }
        display->setText(text);
    }

    void Calculator::backspaceClicked()
    {
        if (waitingForOperand)
            return;

        QString text = display->text();
        text.chop(1);
        if (text.isEmpty()) {
            text = "0";
            waitingForOperand = true;
        }
        display->setText(text);
    }

    void Calculator::clear()
    {
        if (waitingForOperand)
            return;

        display->setText("0");
        waitingForOperand = true;
```

```
}

void Calculator::clearAll()
{
    sumSoFar = 0.0;
    factorSoFar = 0.0;
    pendingAdditiveOperator.clear();
    pendingMultiplicativeOperator.clear();
    display->setText("0");
    waitingForOperand = true;
}

void Calculator::clearMemory()
{
    sumInMemory = 0.0;
}

void Calculator::readMemory()
{
    display->setText(QString::number(sumInMemory));
    waitingForOperand = true;
}

void Calculator::setMemory()
{
    equalClicked();
    sumInMemory = display->text().toDouble();
}

void Calculator::addToMemory()
{
    equalClicked();
    sumInMemory += display->text().toDouble();
}
Button *Calculator::createButton(const QString &text, const char *member)
{
    Button *button = new Button(text);
    connect(button, SIGNAL(clicked()), this, member);
    return button;
}

void Calculator::abortOperation()
{
    clearAll();
    display->setText(tr("####"));
}

bool Calculator::calculate(double rightOperand, const QString &pendingOperator)
{
    if (pendingOperator == tr("+")) {
        sumSoFar += rightOperand;
    } else if (pendingOperator == tr("-")) {
        sumSoFar -= rightOperand;
    } else if (pendingOperator == tr("\303\227")) {
        factorSoFar *= rightOperand;
    } else if (pendingOperator == tr("\303\267")) {
        if (rightOperand == 0.0)
            return false;
        factorSoFar /= rightOperand;
    }
    return true;
}
```

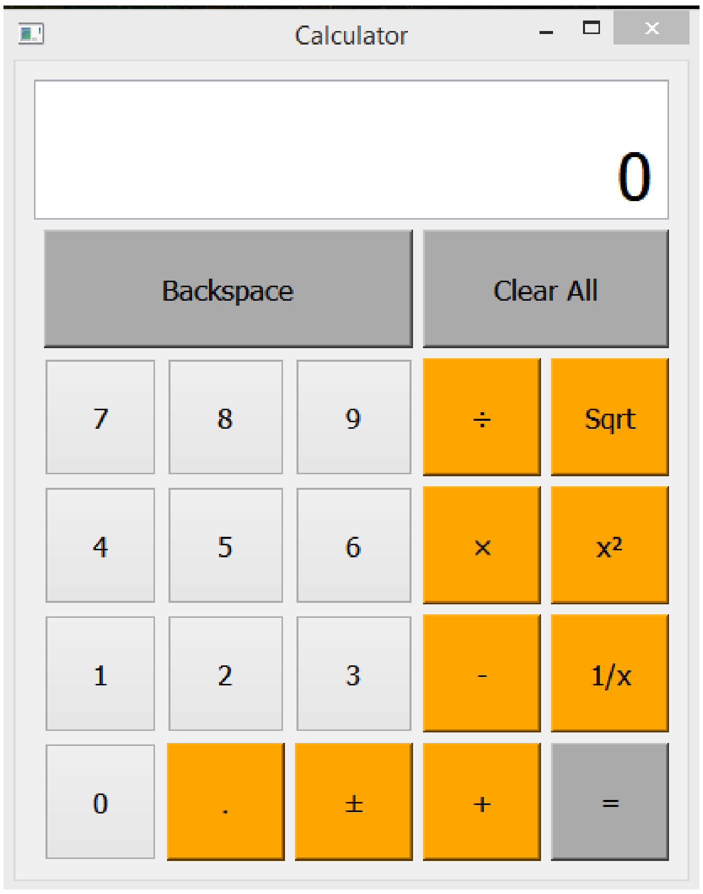
Main.cpp

```
#include <QApplication>

#include "calculator.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    Calculator calc;
    calc.show();
    return app.exec();
}
```

Captură ecran :



Concluzii :

*La realizarea acestei lucrări de laborator am făcut cunoștință cu mediul integrat de dezvoltare Qt astfel am creat un calculator în limbajul C++ care suportă funcțiile de bază + ,-, *, /, putere, radical , schimbarea semnului, backspace, clear.Efectuând această sarcină am studiat detaliat modul de funcționare , clasele și funcțiile oferite de Qt pentru crearea unui calculator cu interfață grafică.*