

1. Цель и задачи

Цель: познакомится с основами работы с графиками и потоками в Qt

Задачи:

* изучить основы работы с QThread;
* изучить основы работы с QChart;
* создать приложение для визуализации сортировки.

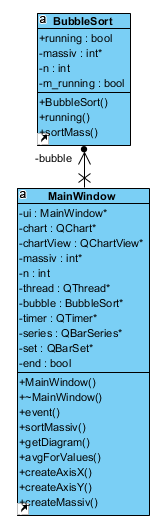
1. Задание

Общее ограничение на лабораторную работу:

* использовать разбитие на файлы основной программы (в main.cpp только функция main.cpp);
* документировать весь код используя аннотации Doxygen.

Задание на лабораторную работу:

1. Эмулируем долгую работу программы над каким-то тяжёлым действием путём сортировки методом пузырька 500.000 тыс. int значений.
2. Требование: программа должна после запуска генерировать массив в 500.000 элементов.
3. Программа должна на основном экране иметь: столбиковую диаграмму (500 элементов), кнопку запуска и кнопку выхода.
4. Программа должна уметь соотносить 500.000 тыс. значений из пункта 2 и 500 столбиков диаграммы (к примеру 1 столбик высчитывает среднее значение 1000 элементов и отображает именно их).
5. При нажатии на кнопку запуска программа должна инициализировать сортировку столбиком.
6. Кнопка выхода из программы должна быть доступна всегда (т. е. сортировка должна идти в отдельном потоке).
7. Раз в 3 секунды должно идти обновление текущего состояния сортируемых данных в график на интерфейсе.
8. ДИАГРАММЫ



*Рисунок 1 – диаграмма классов*

1. КОд программы

Основной выполняемый файл main.cpp:

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

Файл mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

chart = new QChart();

createMassiv();

series = new QBarSeries();

series->append(getDiagram());

chart->addSeries(series);

chart->setTitle("Среднее значение по сортировке");

createAxisX();

createAxisY();

chartView = new QChartView(ui->widget);

chartView->setRenderHint(QPainter::Antialiasing);

chart->legend()->setVisible(0);

chartView->setChart(chart);

chartView->resize(ui->widget->size());

thread=new QThread();

bubble=new BubbleSort(nullptr,massiv,n);

bubble->moveToThread(thread);

timer=new QTimer();

sortMassiv();

connect(ui->start,&QPushButton::clicked,this,&MainWindow::buttons);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::createAxisX()

{

QStringList categories;

for(int i=1;i<=n/1000;i++)

categories.append(QString::number(i));

QBarCategoryAxis \*axisX = new QBarCategoryAxis();

axisX->append(categories);

chart->addAxis(axisX, Qt::AlignBottom);

series->attachAxis(axisX);

}

void MainWindow::createAxisY()

{

QValueAxis \*axisY = new QValueAxis;

axisY->setTitleText("Среднее");

chart->addAxis(axisY, Qt::AlignLeft);

series->attachAxis(axisY);

}

void MainWindow::createMassiv()

{

massiv=new int[n];

QRandomGenerator generator;

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

massiv[i]=generator.bounded(1,1000000);

}

bool MainWindow::event(QEvent \*event)

{

if(event->type()==QEvent::Resize)

chartView->resize(ui->widget->size());

return QMainWindow::event(event);

}

void MainWindow::sortMassiv()

{

connect(ui->start,&QPushButton::clicked,this,&MainWindow::beginSort);

connect(ui->exit,&QPushButton::clicked,this,&MainWindow::endSort);

connect(thread, &QThread::started, bubble, &BubbleSort::run);

connect(timer,&QTimer::timeout,bubble,&BubbleSort::getMassiv, Qt::DirectConnection);

connect(bubble,&BubbleSort::nowMassiv,this,&MainWindow::getMassiv, Qt::DirectConnection);

connect(bubble,&BubbleSort::finished,this,&MainWindow::stopTimer);

connect(bubble,&BubbleSort::finished,bubble,&BubbleSort::getMassiv, Qt::DirectConnection);

connect(bubble,&BubbleSort::finished,this,&MainWindow::resultSort);

}

void MainWindow::beginSort()

{

bubble->setRunning(true);

thread->start();

timer->start(3000);

}

void MainWindow::endSort()

{

if(end==0){

timer->stop();

bubble->setRunning(false);

thread->exit();

thread->wait(4000);

qInfo()<<thread->isRunning();

}

this->close();

}

void MainWindow::getMassiv(int \*mass)

{

massiv=mass;

series->remove(set);

series->insert(0,getDiagram());

}

QBarSet\* MainWindow::getDiagram()

{

set = new QBarSet("This");

int k=0,p=1000,iter=0;

avgForValues(k,p,iter);

QColor col;

col.setRgb(0,0,0);

set->setColor(col);

return set;

}

void MainWindow::avgForValues(int k,int p,int iter)

{

while(p<=n){

int buf=0;

for (int i=k;i<p;i++)

buf+=massiv[i];

buf=buf/1000;

set->insert(iter,buf);

iter++;

k=p;

p+=1000;

}

}

void MainWindow::buttons()

{

ui->start->setDisabled(1);

}

void MainWindow::stopTimer()

{

timer->stop();

qInfo()<<"Timer is:"+QString::number(timer->isActive());

}

void MainWindow::resultSort()

{

thread->exit();

thread->wait(4000);

qInfo()<<thread->isRunning();

end=1;

}

Файл mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QtCharts/QtCharts>

#include<QDebug>

#include<QRandomGenerator>

#include"bubblesort.h"

QT\_CHARTS\_USE\_NAMESPACE

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

/\*\*

\* @brief event Обработка событий

\* @param event Событие

\* @return Событие

\*/

bool event(QEvent \*event) override;

/\*\*

\* @brief sortMassiv Connect для выполнения сортировки

\*/

void sortMassiv();

/\*\*

\* @brief getDiagram Определение множества значений для диаграммы

\* @return Множество

\*/

QBarSet\* getDiagram();

/\*\*

\* @brief avgForValues Вычисление среднего значения

\*/

void avgForValues(int,int,int);

/\*\*

\* @brief createAxisX Создание оси X

\*/

void createAxisX();

/\*\*

\* @brief createAxisY Создание оси Y

\*/

void createAxisY();

/\*\*

\* @brief createMassiv Создание массива

\*/

void createMassiv();

public slots:

/\*\*

\* @brief beginSort Начало сортировки

\*/

void beginSort();

/\*\*

\* @brief endSort Прерывание сортировки

\*/

void endSort();

/\*\*

\* @brief resultSort Окончание сортировки

\*/

void resultSort();

/\*\*

\* @brief getMassiv Получение массива

\* @param mass Массив

\*/

void getMassiv(int\* mass);

/\*\*

\* @brief buttons Блокирование кнопки

\*/

void buttons();

/\*\*

\* @brief stopTimer Остановка таймера

\*/

void stopTimer();

private:

Ui::MainWindow \*ui;

QChart \*chart;

QChartView \*chartView;

int \*massiv;

int n=500000;

QThread \*thread;

BubbleSort \*bubble;

QTimer \*timer;

QBarSeries \*series;

QBarSet \*set;

bool end=0;

};

#endif // MAINWINDOW\_H

Файл bubblesort.cpp:

#include "bubblesort.h"

BubbleSort::BubbleSort(QObject \*parent,int \*mass,int val) : QObject(parent)

{

massiv=mass;

n=val;

}

bool BubbleSort::running() const

{

return m\_running;

}

void BubbleSort::run()

{

while(m\_running){

for(int i = 1; i < n; ++i)

for(int r = 0; r < n-i; r++){

sortMass(r);

if(m\_running==false)

return;

}

setRunning(false); }

emit finished();

return;

}

void BubbleSort::sortMass(int r)

{

if(massiv[r] < massiv[r+1]){

int temp = massiv[r];

massiv[r] = massiv[r+1];

massiv[r+1] = temp;

}

}

void BubbleSort::setRunning(bool running)

{

if (m\_running == running)

return;

m\_running = running;

emit runningChanged(running);

}

void BubbleSort::getMassiv()

{

emit nowMassiv(massiv);

}

Файл bubblesort.h:

#ifndef BUBBLESORT\_H

#define BUBBLESORT\_H

#include <QObject>

#include<QDebug>

class BubbleSort : public QObject

{

Q\_OBJECT

Q\_PROPERTY(bool running READ running WRITE setRunning NOTIFY runningChanged)

public:

explicit BubbleSort(QObject \*parent = nullptr,int \*mass=nullptr,int val=0);

/\*\*

\* @brief running Возврат проверки выполнения сортировки

\* @return Проверка для выполнения сортировки

\*/

bool running() const;

/\*\*

\* @brief sortMass Сортировка пузырьком

\*/

void sortMass(int);

public slots:

/\*\*

\* @brief run Выполнение сортировки

\*/

void run();

/\*\*

\* @brief setRunning Установка проверки выполнения сортировки

\* @param running Проверка для выполнения сортировки

\*/

void setRunning(bool running);

/\*\*

\* @brief getMassiv Получение массива

\*/

void getMassiv();

private:

int \*massiv;

int n;

bool m\_running;

signals:

/\*\*

\* @brief nowMassiv Сигнал для передачи массива

\*/

void nowMassiv(int\*);

/\*\*

\* @brief finished Окончание сортировки

\*/

void finished();

/\*\*

\* @brief runningChanged Сигнал смены значения у проверки для выполнения сортировки

\* @param running Проверка для выполнения сортировки

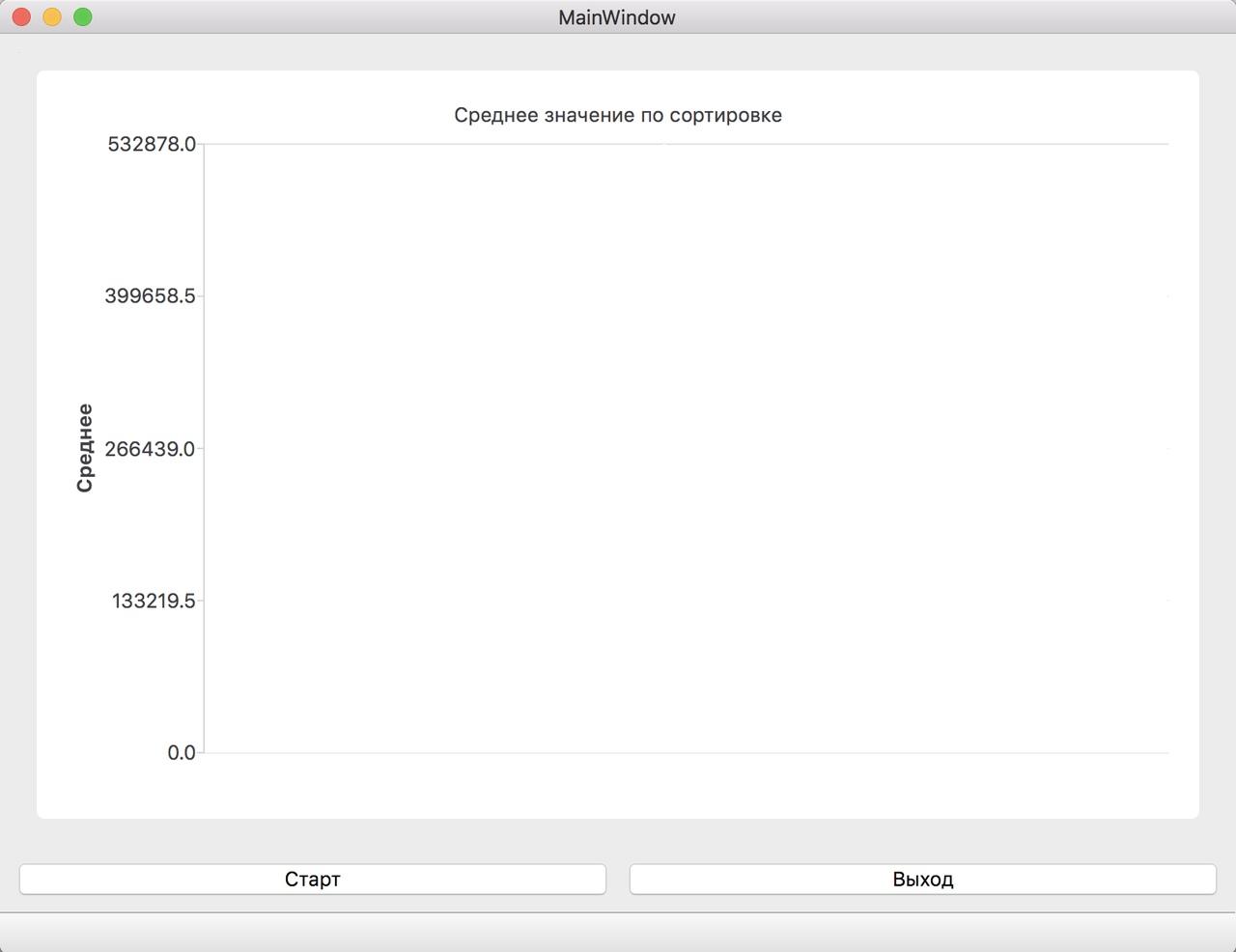
\*/

void runningChanged(bool running);

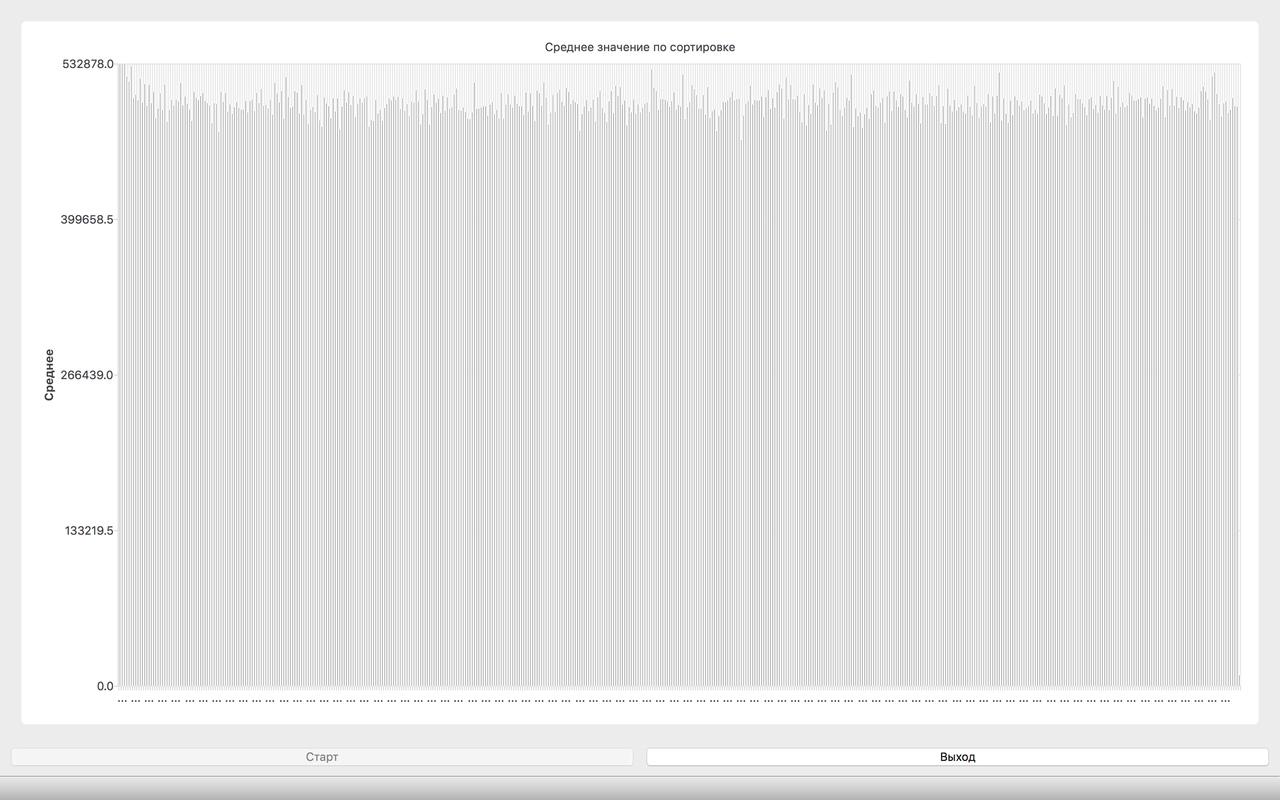
};

#endif // BUBBLESORT\_H

В качестве результата работы программы представлены скриншоты выходных файлов на рисунках 3-4.



*Рисунок 3 – результат работы программы*

**

*Рисунок 4 – результат работы программы*

# ВЫВОД

В процессе выполнения лабораторной работы были получены навыки работы с графиками и потоками в Qt.