**МИНИСТЕРСТВА НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ОБРАЗОВАНИЯ**

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №2

Дисциплина: Компьютерная графика

Тема: «Построение графиков функций»

Выполнила: ст. группы ПВ-31

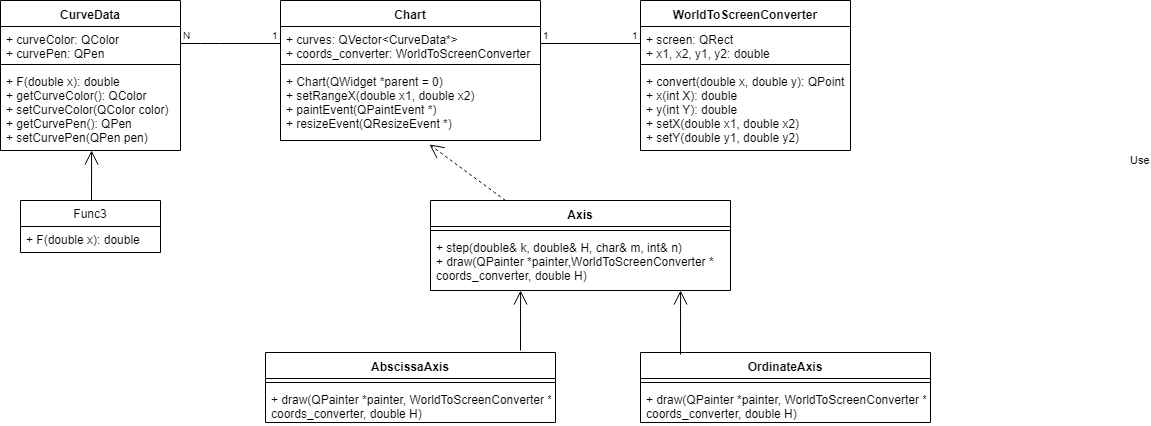
Донцов А.А.

Проверил: Осипов О. В.

Белгород 2019

**Цель работы:** разработка модуля для построения графиков функций одной переменной на декартовой плоскости в среде Qt Creator.

**Диаграмма классов:**

****

**Код программы:**

**chart.h**

#ifndef CHART\_H

#define CHART\_H

#include <QWidget>

#include <QtMath>

#include <QPainter>

#include <QResizeEvent>

#include <QMouseEvent>

#include <QDebug>

#include "windows.h"

class **WorldToScreenConverter**

{

public:

// Экранная система координат

QRect screen;

// Мировая система координат

double x1, x2, y1, y2;

// Функция преобразует координаты точки (x, y) из мировой системы координат в экранную

QPoint **convert**(double x, double y);

// Функция возвращает значение абсциссы в мировой системе координат по её значению X в экранной

double **x**(int X);

// Функция преобразует значение ординаты Y из экранной системы координат в мировую

double **y**(int Y);

// Установить область определения функции

void **setX**(double x1, double x2);

void **setY**(double y1, double y2);

};

//оси

class **Axis**

{

public:

//шаг разметки

void **step**(double& k, double& H, char& m, int& n);

//рисование осей

virtual void ***draw***(QPainter \*painter, WorldToScreenConverter \*coords\_converter, double H) = 0;

};

//ось Ох

class **AbscissaAxis** : public Axis

{

public:

void ***draw***(QPainter \*painter, WorldToScreenConverter \*coords\_converter, double H);

};

//ось Оу

class **OrdinateAxis** : public Axis

{

public:

void ***draw***(QPainter \*painter, WorldToScreenConverter \*coords\_converter, double H);

};

//информация о функции

class **CurveData**

{

// Цвет кривой

QColor curveColor;

// Стиль пера

QPen curvePen;

public:

virtual double ***F***(double x) = 0;

QColor **getCurveColor**();

void **setCurveColor**(QColor color);

QPen **getCurvePen**();

void **setCurvePen**(QPen pen);

};

class **Func3** : public CurveData

{

public:

double ***F***(double x);

};

class **Chart** : public QWidget

{

Q\_OBJECT

public:

explicit **Chart**(QWidget \*parent = 0);

// Массив указателей на структуры, содержащие информацию о функциях f(x)

QVector<CurveData\*> curves;

// Функция устанавливает область определения графика функции

void **setRangeX**(double x1, double x2);

private:

// Объект для преобразования мировых координат в экранные и наоборот

WorldToScreenConverter coords\_converter;

void ***paintEvent***(QPaintEvent \*);

void ***resizeEvent***(QResizeEvent \*);

QPointF mousePos;

void ***mouseMoveEvent***(QMouseEvent \*mouseEvent);

};

#endif // CHART\_H

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~MainWindow();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**chart.cpp**

#include "Chart.h"

QPoint WorldToScreenConverter::**convert**(double x, double y)

{

return QPoint(screen.left() + (screen.right() - screen.left()) / (x2 - x1) \* (x - x1), screen.bottom() - ((screen.bottom() - screen.top()) / (y2 - y1)) \* (y - y1));

}

double WorldToScreenConverter::**x**(int X)

{

return ((X - screen.left()) / ((screen.right() - screen.left()) / (x2 - x1)) + x1);

}

double WorldToScreenConverter::**y**(int Y)

{

return ((screen.bottom() - Y) / ((screen.bottom() - screen.top()) / (y2 - y1)) + y1);

}

void WorldToScreenConverter::**setX**(double x1, double x2)

{

this->x1 = x1;

this->x2 = x2;

}

void WorldToScreenConverter::**setY**(double y1, double y2)

{

this->y1 = y1;

this->y2 = y2;

}

void Axis::**step**(double &k, double &H, char &m, int &n)

{

double p = H / k;

double n1 = log10(p), n2 = log10(p / 2), n5 = log10(p / 5);

double t = qPow(10, ceil(n1)) - p;

double min = p - qPow(10, floor(n1));

m = 1;

n = floor(n1);

if(t < min)

{

m = 1;

n = ceil(n1);

min = t;

}

t = p - 2 \* qPow(10, floor(n2));

if(t < min)

{

m = 2;

n = floor(n2);

min = t;

}

t = 2 \* qPow(10, ceil(n2)) - p;

if(t < min)

{

m = 2;

n = ceil(n2);

min = t;

}

t = p - 5 \* qPow(10, floor(n5));

if(t < min)

{

m = 5;

n = floor(n5);

min = t;

}

t = 5 \* qPow(10, ceil(n5)) - p;

if(t < min)

{

m = 5;

n = ceil(n5);

}

H = k;

k = m \* qPow(10, n); //действительный шаг в мировой

H \*= k;

}

void AbscissaAxis::***draw***(QPainter \*painter, WorldToScreenConverter \*coords\_converter, double H)

{

// k - количество пикселей на одну единицу в мировой системе координат

double k = coords\_converter->screen.width() / (coords\_converter->x2 - coords\_converter->x1);

char m = 0;

int n = 0;

this->step(*k*, *H*, *m*, *n*); //шаг разметки

QPen pen;

pen.setWidth(2);

pen.setColor(Qt::black);

painter->setPen(pen);

QPointF p1 = coords\_converter->convert(coords\_converter->x1, coords\_converter->y1);

p1.setX(p1.x() + pen.width() + 40);

p1.setY(p1.y() - pen.width() - 40);

QPointF p2 = coords\_converter->convert(coords\_converter->x2, coords\_converter->y1);

p2.setX(p2.x() + pen.width() + 40);

p2.setY(p2.y() - pen.width() - 40);

painter->drawLine(p1, p2);

//разметка оси

double start = floor(coords\_converter->x1 / k + 1) \* k;

int j = 0;

for (double i = p1.x() + H; i <= p2.x() - 0.5 \* H; i += H)

{

painter->drawLine(i , p1.y() - 5, i, p1.y() + 5);

painter->drawText(QRectF(QPointF(i - 30, p1.y() + 5), QPointF(i + 30, p1.y() + 25)), Qt::AlignCenter, QString::number(start + (j++) \* k));

}

}

void OrdinateAxis::***draw***(QPainter \*painter, WorldToScreenConverter \*coords\_converter, double H)

{

double k = coords\_converter->screen.height() / (coords\_converter->y2 - coords\_converter->y1);

char m = 0;

int n = 0;

this->step(*k*, *H*, *m*, *n*);

QPen pen;

pen.setWidth(2);

pen.setColor(Qt::black);

painter->setPen(pen);

QPointF p1 = coords\_converter->convert(coords\_converter->x1, coords\_converter->y1);

p1.setX(p1.x() + pen.width() + 40);

p1.setY(p1.y() - pen.width() - 40);

QPointF p2 = coords\_converter->convert(coords\_converter->x1, coords\_converter->y2);

p2.setX(p2.x() + pen.width() + 40);

p2.setY(p2.y() - pen.width() - 40);

painter->drawLine(p1, p2);

//разметка оси

double start = floor(coords\_converter->y1 / k + 1) \* k;

int j = 0;

for (double i = p1.y() - H; i > p2.y() + 0.5 \* H; i -= H)

{

painter->drawLine(p1.x() - 5, i, p1.x() + 5, i);

painter->drawText(QRectF(QPointF(p1.x() - 45, i - 10), QPointF(p1.x() - 5, i + 10)), Qt::AlignCenter, QString::number(start + (j++) \* k));

}

}

QColor CurveData::**getCurveColor**()

{

return curveColor;

}

void CurveData::**setCurveColor**(QColor color)

{

curveColor = color;

}

QPen CurveData::**getCurvePen**()

{

return curvePen;

}

void CurveData::**setCurvePen**(QPen pen)

{

curvePen = pen;

}

double Func3::***F***(double x)

{

return qPow(x, 3);

}

Chart::**Chart**(QWidget \*parent) : QWidget(parent)

{

}

void Chart::***paintEvent***(QPaintEvent \*)

{

QPainter painter(this);

AbscissaAxis Ox;

OrdinateAxis Oy;

Ox.*draw*(&painter, &coords\_converter, 100);

Oy.*draw*(&painter, &coords\_converter, 100);

this->curves.push\_back(new Func3);

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

{

curves[i]->setCurveColor(QColor(255, 0, 0, 255));

QPen pen;

pen.setWidth(2);

pen.setColor(curves[i]->getCurveColor());

painter.setPen(pen);

QPoint t1 = coords\_converter.convert(coords\_converter.x1, curves[i]->*F*(coords\_converter.x1));

t1.setX(t1.x() + pen.width() + 40);

t1.setY(t1.y() - pen.width() - 40);

QPoint t2;

for (double j = coords\_converter.x1; j <= coords\_converter.x2; j += 0.001)

{

t2 = coords\_converter.convert(j, curves[i]->*F*(j));

t2.setX(t2.x() + pen.width() + 40);

t2.setY(t2.y() - pen.width() - 40);

painter.drawLine(t1, t2);

t1 = t2;

}

}

//изменение размера окна

if (QWidget::width() > QWidget::parentWidget()->width() - 60)

QWidget::resize(QWidget::parentWidget()->width() - 50, QWidget::height());

else if (QWidget::width() < QWidget::parentWidget()->width() - 90)

QWidget::resize(QWidget::parentWidget()->width() - 50, QWidget::height());

if (QWidget::height() > QWidget::parentWidget()->height())

QWidget::resize(QWidget::width(), QWidget::parentWidget()->height());

else if (QWidget::height() < QWidget::parentWidget()->height() + 1)

QWidget::resize(QWidget::width(), QWidget::parentWidget()->height());

//координаты около курсора

QPen pen\_for\_coords;

pen\_for\_coords.setColor(QColor(60, 62, 104, 255));

painter.setPen(pen\_for\_coords);

QPointF mousePosWorld = QPointF(coords\_converter.x(mousePos.x()), coords\_converter.y(mousePos.y()));

QString str = QString::number(mousePosWorld.x()) + ": " + QString::number(mousePosWorld.y());

painter.drawText(QRectF(mousePos.x(), mousePos.y() - 10, 10000, 30), Qt::AlignLeft, str);

}

void Chart::***resizeEvent***(QResizeEvent \*)

{

coords\_converter.screen = QWidget::geometry();

coords\_converter.setX(-7.538, -7.5);

coords\_converter.setY(-428.7, -421.8);

}

void Chart::***mouseMoveEvent***(QMouseEvent \*mouseEvent)

{

mousePos = mouseEvent->pos();

}

**mainwindow.cpp**

#include "MainWindow.h"

#include "ui\_mainwindow.h"

#include <QTimer>

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

QTimer \*timer = new QTimer(this);

connect(timer, SIGNAL(timeout()), this, SLOT(update()));

timer->start(1);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

**main.cpp**

#include "MainWindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

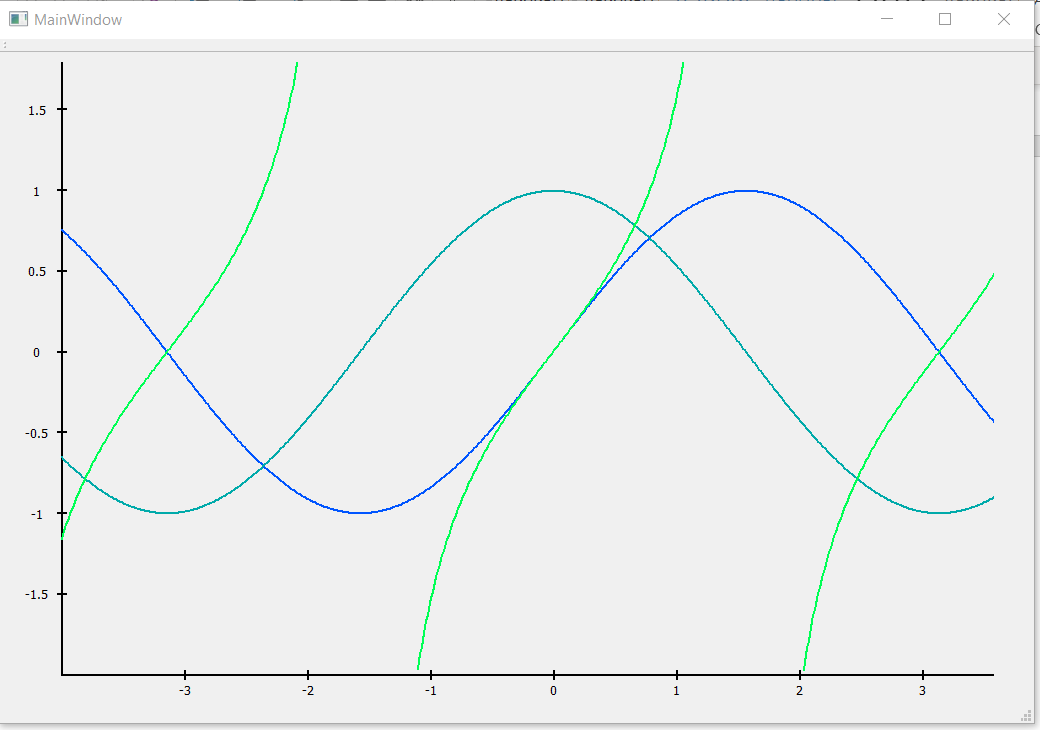
MainWindow w;

w.show();

return a.exec();

}

**Результат работы программы:**

****