МИНИСТЕРСТВО 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

по дисциплине: Компьютерная графика

тема: «Создание графического компонента для построения графиков функций»

Выполнил: ст. группы ВТ-32

Воскобойников И. С.

Проверил: Гибкин Ю. С.

Белгород 2020 г.

Цель работы: Разработка на базе класса QWidget собственного компонента для построения графика функций.

Задания для выполнения к работе

1. Разработать классы для перевода координат из мировой в экранную системы и наоборот; классы для работы с осями абсцисс и ординат.
2. Написать программу для построения графика произвольной функции на интервале от а до b . a и b задается пользователем. Область значений должна вычисляться автоматически.

Main.c

#include "graph.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

Graph w;

w.show();

return a.exec();

}

Axic.h

#ifndef AXIS\_H

#define AXIS\_H

#include "worldtoscreenconverter.h"

#include <QPainter>

class Axis

{

public:

void step(double \*k, int h, int \*m, int \*n);

//прорисоква осей Ox и Oy

void draw(QPainter \*painter, int h, WorldToScreenConverter conv);

};

class AbscissaAxis : Axis

{

public:

void draw(QPainter \*painter, int h, WorldToScreenConverter conv);

};

class OrdinateAxis : Axis

{

public:

void draw(QPainter \*painter, int h, WorldToScreenConverter conv);

};

#endif // AXIS\_H

Axic.c

#include "axis.h"

#include "worldtoscreenconverter.h"

void Axis::step(double \*k, int h, int \*m, int \*n){

double h\_m = h/(\*k); // 1/k = отношение ширины в экранных координатах к мировым

double n1 = log10(h\_m); //для решения уравнение h\_m = 10 ^ n1

double n2 = log10(h\_m/2); //для решения уравнение h\_m = 2 \* 10 ^ n2

double n5 = log10(h\_m/5); //для решения уравнение h\_m = 5 \* 10 ^ n5

double min = pow(10, ceil(n1)) - h\_m;//ceil - округление вверх

\*k = min + h\_m;

\*m = 1;

\*n = ceil(n1);

double current = pow(10, floor(n1));

if (h\_m - current < min){//floor - округление вниз

min = h\_m - current;

\*k = current;

\*n = floor(n1);

}

}

//прорисоква осей Ox и Oy

void Axis::draw(QPainter \*painter, int h, WorldToScreenConverter conv){

QPen pen(Qt::black,3,Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);

painter->setPen(pen);

if (conv.getXMin() <= 0 && 0 <= conv.getXMax()) //ось Оу

//если область определения в отрицательной области, рисуем в правой части экрана

painter->drawLine(conv.convert(0,conv.getYMin()), conv.convert(0, conv.getYMax()));

else if (conv.getXMin() < conv.getXMax()){

if (conv.getXMax() < 0)

painter->drawLine(conv.getBottomRight(), conv.getTopRight());

else

painter->drawLine(conv.getBottomLeft(), conv.getTopLeft());

}

if (conv.getYMin() <= 0 && 0 <= conv.getYMax()) //ось Оx

//если область значений в отрицательной части, русем наверху

painter->drawLine(conv.convert(conv.getXMin(), 0), conv.convert(conv.getXMax(), 0));

else if (conv.getYMin() < conv.getYMax()){

if (conv.getYMax() < 0)

painter->drawLine(conv.getTopLeft(), conv.getTopRight());

else

painter->drawLine(conv.getBottomLeft(), conv.getBottomRight());

}

painter->setPen(QPen(Qt::blue,1,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));

}

void OrdinateAxis::draw(QPainter \*painter, int h, WorldToScreenConverter conv){

int m, n;

double k = conv.getHeight()/(conv.getYMax() - conv.getYMin());

step(&k, h, &m, &n);

double k1 = ceil(conv.getYMin()/k)\*k;

for (; k1 <= conv.getYMax(); k1 += k){

//прорисовка сетки с шагом k

painter->setPen(Qt::gray);

painter->drawLine(conv.convert(conv.getXMin(), k1), conv.convert(conv.getXMax(), k1));

painter->setPen(Qt::black);

if (fabs(k1) - 1.e-14 < 0){

painter->drawText(QPoint(conv.getLeft()-10, conv.Yscr(k1)+5), "0");

}

else{

QString str;

if(fabs(k1) < 10)

str = QString::number(k1, 'f', 2);

else

str = QString::number(k1, 'e', 2);

painter->drawText(QPoint(conv.getLeft()-60, conv.Yscr(k1)+5), str);

}

}

painter->setPen(QPen(Qt::blue,1,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));

}

void AbscissaAxis::draw(QPainter \*painter, int h, WorldToScreenConverter conv){

int m, n;

double k = conv.getWidth()/(conv.getXMax() - conv.getXMin());

step(&k, h, &m, &n);

double k1 = ceil(conv.getXMin()/k)\*k;

for (; k1 <= conv.getXMax(); k1 += k){

painter->setPen(Qt::gray);

painter->drawLine(conv.convert(k1, conv.getYMin()), conv.convert(k1, conv.getYMax()));

painter->setPen(Qt::black);

QString str;

if(fabs(k1) < 10 && fabs(k1) > 0.1)

str = QString::number(k1, 'f', 2);

else

str = QString::number(k1, 'e', 2);

QPoint p(conv.Xscr(k1)-20, conv.getBottom()+18);

painter->translate(p);

painter->rotate(30);

painter->drawText(0, 0, str);

painter->rotate(-30);

painter->translate(-p);

}

}

curveddata.h

#ifndef CURVEDDATA\_H

#define CURVEDDATA\_H

#include <QColor>

#include <QPen>

#include <qpainter.h>

class CurveData

{

public:

CurveData();

QColor curveColor; // Цвет кривой

QPen curvePen; // Стиль пера

virtual double *F*(double x); // Функция для рисования

void setColor(QColor color);

};

class Function1 : public CurveData

{

public:

Function1();

double *F*(double x); // Функция для рисования

};

class Function2 : public CurveData

{

public:

Function2();

double *F*(double x); // Функция для рисования

};

class Function3 : public CurveData

{

public:

Function3();

double *F*(double x); // Функция для рисования

};

#endif // CURVEDDATA\_H

curveddata.c

#include "curveddata.h"

CurveData :: CurveData(){

QColor(Qt::blue);

QPen curvePen(Qt::blue, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);

}

double CurveData :: *F*(double x) { return 1; }

void CurveData :: setColor(QColor color){

curveColor = color;

curvePen = QPen(color, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);

}

Function1 :: Function1(){}

double Function1 :: *F*(double x) { return sin(x)\*x\*x + x; }

Function2 :: Function2(){}

double Function2 :: *F*(double x) { return cos(x) + x\*x - x\*2; }

Function3 :: Function3(){}

double Function3 :: *F*(double x) { return x\*x/5 + 10 ;}

worldtoscreenconverter.h

#ifndef WORLDTOSCREENCONVERTER\_H

#define WORLDTOSCREENCONVERTER\_H

#include <QRect>

#include <QPoint>

class WorldToScreenConverter

{

public:

WorldToScreenConverter(QPoint minScr, QPoint maxScr, double xMinWrld, double yMinWrld, double xMaxWrld, double yMaxWrld);

QRect Screen;

double xW\_min, xW\_max, yW\_min, yW\_max;

double xxx, yyy, xw, yh;

QPoint convert(double xWorld, double yWorld);

int Xscr(double xWorld);

int Yscr(double yWorld);

double Xwrld(int xScreen);

double Ywrld(int yScreen);

int getWidth();

int getHeight();

QPoint getTopLeft();

QPoint getTopRight();

QPoint getBottomLeft();

QPoint getBottomRight();

double getXMin();

double getXMax();

double getYMin();

double getYMax();

int getLeft();

int getBottom();

};

#endif // WORLDTOSCREENCONVERTER\_H

worldtoscreenconverter.c

#include "worldtoscreenconverter.h"

WorldToScreenConverter::WorldToScreenConverter(QPoint minScr, QPoint maxScr, double xMinWrld, double yMinWrld, double xMaxWrld, double yMaxWrld):

Screen(QRect(minScr, maxScr)),

xW\_min(xMinWrld),

yW\_min(yMinWrld),

xW\_max(xMaxWrld),

yW\_max(yMaxWrld)

{

xxx = Screen.width()/(xW\_max-xW\_min);

yyy = Screen.height()/(yW\_max-yW\_min);

xw = (xW\_max-xW\_min)/Screen.width();

yh = (yW\_max-yW\_min)/Screen.height();

}

QPoint WorldToScreenConverter::convert(double xWorld, double yWorld){

int x\_local = Screen.left() + (int)((xWorld-xW\_min)\*xxx);

int y\_local = 1 + Screen.bottom() - (int)((yWorld-yW\_min)\*yyy);

return QPoint(x\_local, y\_local);

}

int WorldToScreenConverter::Xscr(double xWorld){

return Screen.left() + (int)((xWorld-xW\_min)\*xxx);

}

int WorldToScreenConverter::Yscr(double yWorld){

return 1 + Screen.bottom() - (int)((yWorld-yW\_min)\*yyy);

}

double WorldToScreenConverter::Xwrld(int xScreen){

return ((xScreen-Screen.x())\*xw);

}

double WorldToScreenConverter::Ywrld(int yScreen){

return ((Screen.y()+Screen.height()-yScreen)\*yh);

}

int WorldToScreenConverter::getWidth(){

return Screen.width();

}

int WorldToScreenConverter::getHeight(){

return Screen.height();

}

QPoint WorldToScreenConverter::getTopLeft(){

return Screen.topLeft();

}

QPoint WorldToScreenConverter::getTopRight(){

return Screen.topRight();

}

QPoint WorldToScreenConverter::getBottomLeft(){

return Screen.bottomLeft();

}

QPoint WorldToScreenConverter::getBottomRight(){

return Screen.bottomRight();

}

double WorldToScreenConverter::getXMin(){

return xW\_min;

}

double WorldToScreenConverter::getXMax(){

return xW\_max;

}

double WorldToScreenConverter::getYMin(){

return yW\_min;

}

double WorldToScreenConverter::getYMax(){

return yW\_max;

}

int WorldToScreenConverter::getLeft(){

return Screen.left();

}

int WorldToScreenConverter::getBottom(){

return Screen.bottom();

}

Graph.h

#ifndef GRAPH\_H

#define GRAPH\_H

#include <QMainWindow>

#include "curveddata.h"

namespace Ui {

class Graph;

}

class Graph : public QMainWindow

{

Q\_OBJECT

public:

explicit Graph(QWidget \*parent = 0);

~*Graph*();

double x\_min=0, x\_max=1, y\_min=0, y\_max=1;

/\*задание функции\*/

QVector<CurveData\*> curves;

/\*вычисление y\_min и y\_max\*/

void calcYArea();

void *paintEvent*(QPaintEvent \*);

private slots:

void on\_lineEdit\_editingFinished();

void on\_lineEdit\_2\_editingFinished();

private:

Ui::Graph \*ui;

};

#endif // GRAPH\_H

Graph.c

#include "graph.h"

#include "ui\_graph.h"

#include <qpainter.h>

#include <QWidget>

#include <cfloat>

#include "axis.h"

Graph::Graph(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::Graph)

{

ui->setupUi(this);

Function1 \*f1 = new Function1();

Function2 \*f2 = new Function2();

Function3 \*f3 = new Function3();

f1->setColor(Qt::red);

f2->setColor(Qt::blue);

f3->setColor(Qt::green);

curves.insert(0, f1);

curves.insert(1, f2);

curves.insert(2, f3);

calcYArea();

}

Graph::~*Graph*()

{

delete ui;

}

/\*вычисление y\_min и y\_max\*/

void Graph::calcYArea(){

double h = (x\_max - x\_min) / width();

double x = x\_min;

y\_min = curves[0]->*F*(x);

y\_max = y\_min;

QVector<CurveData\*>::iterator cur;

for(cur = curves.begin(); cur != curves.end(); cur++){

x = x\_min;

for (; x <= x\_max; x+=h){

double y = (\*cur)->F(x);

if (y\_min > y)

y\_min = y;

if (y\_max < y)

y\_max = y;

}

}

}

void Graph::*paintEvent*(QPaintEvent \*){

//настройка рисователя

QPainter paint(this);

QPoint minScreen(70, 60);

QPoint maxScreen(width() - 30, height() - 60);

WorldToScreenConverter converter(minScreen, maxScreen, x\_min, y\_min, x\_max, y\_max);

int h = 70;

/\*построение графика\*/

//вычисление шага

double h\_w = (x\_max - x\_min) / converter.getWidth();

double x = x\_min;

//инифицлизация начальной точки

QVector<CurveData\*>::iterator cur;

for(cur = curves.begin(); cur != curves.end(); cur++) {

paint.setPen((\*cur)->curvePen);

x = x\_min;

QPoint p1 = converter.convert(x, (\*cur)->F(x));

for (x = x + h\_w; x < x\_max; x += h\_w){

QPoint p2 = converter.convert(x, (\*cur)->F(x));

//рисование хорды

paint.drawLine(p1, p2);

p1 = p2;

}

}

OrdinateAxis oy = OrdinateAxis();

oy.draw(&paint, h, converter); //отрисовка сетки для оси Oy

AbscissaAxis ox = AbscissaAxis();

ox.draw(&paint, h, converter); //отрисовка сетки для оси Ox

}

void Graph::on\_lineEdit\_editingFinished()

{

x\_min = ui->lineEdit->text().toDouble();

calcYArea();

if (x\_min < x\_max)

repaint();

}

void Graph::on\_lineEdit\_2\_editingFinished()

{

x\_max = ui->lineEdit\_2->text().toDouble();

calcYArea();

if (x\_min < x\_max)

repaint();

}

