**MainWindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QLineEdit>

#include <QLabel>

#include "roots.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private:

Ui::MainWindow \*ui;

const int max = 7;

QLineEdit\* lines[8];

QLabel\* labels[8];

Roots\* solution;

private slots:

void setLevel(int level);

void sliderChanged(int value);

void getRoots();

};

#endif // MAINWINDOW\_H

**MainWindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

this->setWindowTitle( "Roots" );

this->setFixedSize(435, 295);

ui->spinBox->setMinimum(1);

ui->spinBox->setMaximum(max);

ui->output->setReadOnly(true); // поле вывода только для чтения

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

{

lines[i] = NULL;

labels[i] = NULL;

}

connect(ui->spinBox, SIGNAL(valueChanged(int)), this, SLOT(setLevel(int)));

connect(ui->slider, SIGNAL(valueChanged(int)), this, SLOT(sliderChanged(int)));

connect(ui->but, SIGNAL(clicked(bool)), this, SLOT(getRoots()));

emit ui->spinBox->valueChanged(1);

}

void MainWindow::setLevel(int level)

{

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

if (lines[i] != NULL)

{

delete(lines[i]);

lines[i] = NULL;

delete(labels[i]);

labels[i] = NULL;

}

QRegExp reg("[-]?[0-9]{1,4}"); // регулярные выражения

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

{

// lines

lines[i] = new QLineEdit(this);

lines[i]->setFixedSize(40,20);

lines[i]->setValidator(new QRegExpValidator(reg,this));

ui->lay->addWidget(lines[i], 0, i);

// label

labels[i] = new QLabel(this);

QString str = "a" + QString::number(level - i);

labels[i]->setFixedSize(40, 10);

labels[i]->setText(str);

ui->lay->addWidget(labels[i], 1, i);

}

}

void MainWindow::sliderChanged(int value)

{

ui->signs->setText(QString::number(value));

}

void MainWindow::getRoots()

{

//-----------------получить коэффициенты---------------

int level = ui->spinBox->value(); // получить значение старшего коэф

int a[max + 1]; // массив коэф

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

{

bool ok; // useless

QString str = lines[i]->text();

a[level - i] = str.toInt(&ok);

}

//--------------------найти корни--------------------

Roots solution(a, level); // массив коэф и старшую степень

bool check = ui->checkBox->isChecked(); // проверяем галочку

float\* roots = solution.getRoots(check); // получаем корни

int num\_roots = solution.getNumRoots(); // и их кол-во

QString message = "Найдено " + QString::number(num\_roots) + " корень/ей";

ui->statusBar->showMessage(message);

//--------------------вывести корни-----------------

// очистить поле вывода

ui->output->clear();

int k = ui->slider->value(); // количество знаков после запятой

for (int i = 0; i < num\_roots; i++)

{

QString str = QString::number(roots[i]);

int index\_point = str.indexOf(".", 1);

if (index\_point > 0) // если запятая вообще есть

str.remove(index\_point + 1 + k, 10);

ui->output->insertPlainText(str+"\n"); // вставить текст в поле вывода

}

delete [] roots;

}

MainWindow::~*MainWindow*()

{

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

if (lines[i] != NULL)

{

delete lines[i];

delete labels[i];

}

delete ui;

}

**Roots.h**

#ifndef ROOTS\_H

#define ROOTS\_H

#pragma once

#include <cmath>

#include <iostream>

using std::cout;

class Roots

{

private:

float\* a; // коэффициенты

int a0; // ненулевой коэф при младшей степени

int an; // старший коэф.

int n; // старшая степень

float f(float x); // сама функция

bool checkRoot(float x);

float\* roots; // массив корней

int numRoots; // количество корней

public:

Roots(int mas[], int n);

float\* getRoots(bool onlyRa);

int getNumRoots()

{

return numRoots;

}

~Roots();

};

#endif // ROOTS\_H

**Roots.cpp**

#include "Roots.h"

Roots::Roots(int mas[], int k)

{

n = k;

// младшая степень (м.б. нулевая!)

int z = 0;

while (mas[z] == 0)

z++;

a0 = abs(mas[z]);

// старшая степень

an = abs(mas[n]);

a = new float[n + 1];

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

a[i] = mas[i];

roots = new float[n];

numRoots = 0;

}

float Roots::f(float x)

{

float res(0);

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

res += a[i] \* pow(x, i);

res = floor(res \* 1000) / 1000; // округление

return res;

}

float\* Roots::getRoots(bool onlyRa)

{

//---------------проверяем рациональные корни-------------

// проверить на корни +-1 и 0

while (checkRoot(1));

while (checkRoot(-1));

while (checkRoot(0));

for (int p = 1; p <= a0 && n > 0; p++)

if (a0 % p == 0)

{

for (int q = 1; q <= an && n > 0; q++)

if (an % q == 0 && p / float(q) != 1)

{

while (checkRoot(p / float(q)));

while (checkRoot(-p / float(q)));

}

}

if (n == 0)

return roots;

if (onlyRa)

return roots;

//------------проверяем иррациональные корни---------------

//------------метод половинного деления--------------------

const int t1 = -10; // границы проверки корней

const int t2 = 10;

for (float i = t1; i < t2; i += 0.5)

if (f(i) \* f(i + 0.5) < 0) // разных знаков

{

float a = i;

float b = i + 0.5;

float c = (a + b) / 2;

while (f(a) != 0 && f(b) != 0)

{

(f(a) \* f(c) < 0) ? b = c : a = c; // изменяем границы

c = (a + b) / 2;

}

checkRoot(a);

checkRoot(b);

}

return roots;

}

bool Roots::checkRoot(float x)

{

// проверка на дурака

if (n == 0)

return false;

float res = f(x);

if (res != 0)

return false;

else

{

// понизить степень, разделив многочлен на корень

// по схеме горнера

float\* b = new float[n];

b[n - 1] = a[n];

for (int i = n - 1; i > 0; i--)

{

float d = a[i] + x \* b[i];

b[i - 1] = d;

}

delete[] a;

a = b;

roots[numRoots] = x;

numRoots++;

n--;

return true;

}

}

Roots::~Roots()

{

// NO delete[] roots!!!

delete[] a;

}