МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедры МО ЭВМ

Курсовая работа
по дисциплине «Программирование»
Тема: работа с .bmp файлами

Студент гр. 8304 __ Птухов Д.А.

Преподаватель __ Чайка К.В

Санкт-Петербург

2019

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Птухов Д. А.

Группа 8304

Тема работы: работа с .bmp файлами

Исходные данные:

Написать программу для обработки .bmp файлов с GUI, создать собственный класс для хранения изображения и реализовать методы: рисование квадрата или круга, rgb-компонент и поворот изображения (части изображения).

Содержание пояснительной записки:

- Содержание
- Введение
- MainWindow
- Окна для работы с подзадачами
- Image
- Rotate
- Painter
- Тестирование
- Исходный код
- Использованные источники

Дата выдачи задания: 07.03.2019

Дата защиты реферата:

Студент _____ Птухов Д. А.

Преподаватель _____ Чайка К. В.

АННОТАЦИЯ

В данной работе была создана программа, являющаяся desktop-приложением для работы с файлами-изображениями формата bmp. Был разработан GUI для загрузки/сохранения файлов, рисования круга или квадрата, создания rgb-компонента и поворота изображения или его части. Проведена работа по форматированию кода и предоставлено тестирование программы.

SUMMARY

In this paper, a program was created that is a desktop application for working with image files in bmp format. A GUI was developed to load / save files, draw a circle or square, create an rgb component and rotate an image or part of it. Work was done on formatting the code and provided testing program. Work was done on formatting the code and provided testing program.

Оглавление

Введение.

Приложение написано на языке C++ с использованием фреймворка Qt. Для работы с .bmp был создан класс Image, алгоритмы для работы с bmp были написаны без использования средств Qt.

Цель и условие работы.

Вариант 9

Общие сведения

- 24 бита на цвет
- без сжатия
- файл всегда соответствует формату BMP (но стоит помнить, что версий у формата несколько)
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна реализовывать весь следующий функционал по обработке bmp-файла

1. Рисование квадрата с диагоналями. Квадрат определяется:
    - Координатами левого верхнего угла
    - Размером стороны
    - Толщиной линий
    - Цветом линий
    - Может быть залит или нет (диагонали располагаются “поверх” заливки)
    - Цветом которым он залит, если пользователем выбран залитый
2. Фильтр rgb-компонент. Этот инструмент должен позволять для всего изображения либо установить в 0 либо установить в 255 значение заданной компоненты. Функционал определяется
    - Какую компоненту требуется изменить
    - В какой значение ее требуется изменить
3. Поворот изображения (части) на 90/180/270 градусов. Функционал определяется
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
    - Углом поворота
4. Рисование окружности. Окружность определяется:
    - **либо** координатами левого верхнего и правого нижнего угла квадрата, в который она вписана, **либо** координатами ее центра и радиусом
    - толщиной линии окружности
    - цветом линии окружности
    - окружность может быть залитой или нет
    - цветом которым залита сама окружность, если пользователем выбрана залитая окружность

1. MainWindow.

В функции main() создается экземпляр класса MainWindow. Далее запускается метод exec().

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    app.setWindowIcon(QIcon("D:/qt/course work/coursework/Icons/paint.png"));
    MainWindow w;
    w.setWindowTitle("Course work");
    w.show();
    return app.exec();
}
```

Класс MainWindow является основообразующим классом в данном проекте. Он предназначается для создания общего интерфейса программы, взаимодействия с классом Dialog, который предназначается для создания диалоговых окон и передаче данных из них в класс Painter, а также для работы с кнопками, созданными с .ui файле.

Конструктор класса MainWindow, а также функции для работы с интерфейсом.

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    label = new DragableFrame;
    CreateCircleMenu();
    CreateRotateMenu();
    SetIcons();
}

MainWindow::~MainWindow()
{
    delete ui;
    delete good_filename;
}

void MainWindow::CreateCircleMenu()
{
    QMenu* menu = new QMenu;
    QAction* bt1 = new QAction("Radius and center coordinates");
    QAction* bt2 = new QAction("Square coordinates");
    bt1->setIcon(QIcon("D:/qt/course work/coursework/Icons/center_circle.png"));
    bt2->setIcon(QIcon("D:/qt/course work/coursework/Icons/square_and_circle.png"));
    menu->addAction(bt1);
    menu->addAction(bt2);
    connect(bt1, SIGNAL(triggered()), this,
SLOT(on_actionRadius_and_center_coordinates_triggered()));
    connect(bt2, SIGNAL(triggered()), this,
SLOT(on_actionSquare_coordinates_triggered()));
    ui->actionCircle_2->setMenu(menu);
}

void MainWindow::CreateRotateMenu()
{
    QMenu* menu = new QMenu;
    QAction* bt1 = new QAction("Full picture");
    QAction* bt2 = new QAction("Part of picture");
    bt1->setIcon(QIcon("D:/qt/course work/coursework/Icons/rotate_full.png"));
```

```
    bt2->setIcon(QIcon("D:/qt/course work/coursework/Icons/rotate_part.png"));
    menu->addAction(bt1);
    menu->addAction(bt2);
    connect(bt1, SIGNAL(triggered()), this, SLOT(on_RotateFullPicture_triggered()));
    connect(bt2, SIGNAL(triggered()), this,
SLOT(on_RotatePartOfPicture_triggered()));
    ui->actionRotate->setMenu(menu);
}

void MainWindow::SetIcons()
{
    ui->actionRgb->setIcon(QIcon("D:/qt/course work/coursework/Icons/rgb.png"));
    ui->actionCircle_2->setIcon(QIcon("D:/qt/course
work/coursework/Icons/circle.png"));
    ui->actionSquare->setIcon(QIcon("D:/qt/course
work/coursework/Icons/square.png"));
    ui->actionStep_back_2->setIcon(QIcon("D:/qt/course
work/coursework/Icons/step_back.png"));
    ui->actionRotate->setIcon(QIcon("D:/qt/course
work/coursework/Icons/rotate.png"));
    ui->actionFill_color->setIcon(QIcon("D:/qt/course
work/coursework/Icons/fill_color.jpg"));
    ui->actionLine_thickness->setIcon(QIcon("D:/qt/course
work/coursework/Icons/thickness.png"));
    ui->actionChoosing_color->setIcon(QIcon("D:/qt/course
work/coursework/Icons/line_color.jpg"));
    ui->actionOpen->setIcon(QIcon("D:/qt/course
work/coursework/Icons/open_file.png"));
    ui->actionSave->setIcon(QIcon("D:/qt/course
work/coursework/Icons/save_file.png"));
    ui->actionCreate->setIcon(QIcon("D:/qt/course
work/coursework/Icons/create_file.png"));
    ui->actionInfo_about_File->setIcon(QIcon("D:/qt/course
work/coursework/Icons/file_info.png"));
    ui->actionAbout_program_2->setIcon(QIcon("D:/qt/course
work/coursework/Icons/info.png"))
```

Класс также содержит слоты, реализующие меню (открытие/закрытие, создание и сохранение файла, вывод справочной информации). Для открытия файла используется QFileDialog, для создания нового файла и сохранения используются методы Image.

```
void MainWindow::on_actionOpen_triggered()
{
    filename = QFileDialog::getOpenFileName(this, "Choose File",
tr("C:/Users/denis/Desktop"), tr("*.bmp"));
    if (!filename.isEmpty() && !LoadFile(filename))
    {
        QMessageBox::StandardButton reply = QMessageBox::question(this, "Info",
                                                                  "Do you want to
try again?");
        if (reply == QMessageBox::Yes) on_actionOpen_triggered();
    }
}

int MainWindow::LoadFile(QString filename)
{
    good_filename = new char[filename.length() + 1];
    std::strcpy(good_filename, filename.toLatin1().constData());

    int f = label->getImg()->load(good_filename);
    if (f <= 0)
```

```
    {
        QMessageBox::warning(this, "Error", "I can't open this file or his type is
uncorrect!");
        return 0;
    }

    label->resize(label->getImg()->getWidth(), label->getImg()->getHeight());
    label->getPnt()->LoadSize(label->getImg()->getWidth(), label->getImg()-
>getHeight());
    label->setPixmap(label->getImg()->toQPixmap(1));
    Scroll();

    return 1;
}

void MainWindow::Scroll()
{
    label->setBackgroundRole(QPalette::Base);
    label->setSizePolicy(QSizePolicy::Ignored, QSizePolicy::Ignored);
    label->setScaledContents(true);

    scrollArea = new QScrollArea;
    scrollArea->setWidget(label);
    setCentralWidget(scrollArea);
}

void MainWindow::on_actionCreate_triggered()
{
    label->getImg()->create();
    label->setPixmap(label->getImg()->toQPixmap(1));
    label->getPnt()->LoadSize(label->getImg()->getWidth(), label->getImg()-
>getHeight());
}

void MainWindow::on_actionSave_triggered()
{
    QString fileName = QFileDialog::getSaveFileName(this, "Сохранить как...", "./",
"*.bmp");
    if (fileName.isNull()) return;
    std::strcpy(good_filename, fileName.toLatin1().constData());
    for (unsigned int i = 0; i < std::strlen(good_filename); i++) if
(good_filename[i] == '\\') good_filename[i] = '/';
    label->getImg()->save(good_filename);
}

void MainWindow::on_actionAbout_program_2_triggered()
{
    QMessageBox::information(this, "Info", "This program was created by Denis Ptuhov
using the QtCreator(v. 4.8.2)\n"
                                           "HotKeys:\n"
                                           "\t1) Ctrl + O - Open File\n"
                                           "\t2) Ctrl + S - Save File\n"
                                           "\t3) Ctrl + I - Information about
file\n"
                                           "\t4) Ctrl + R - Create File");
}

void MainWindow::on_actionInfo_about_File_triggered()
{
    QMessageBox::information(this, "File info", label->getImg()->info());
}
```

В данном классе (как и во всех далее) в деструкторе происходит освобождение выделенной памяти.

2. DragableFrame.

Это один из основных классов данного проекта в котором хранятся используемые объекты классов Image и Painter. Они отвечают за хранение изображения и рисования на нем соответственно. Также при помощи метода setPixmap() было загружено изображение, содержащееся в классе Image. Методы ActivateDoing…() отвечают за активацию отлавливания кликов мыши при помощи метода QMousePressEvent, а в дальнейшем за требуемое изменение изображения.

```
DragableFrame::DragableFrame()
{
}

void DragableFrame::mousePressEvent(QMouseEvent *event)
{
    if (circle)
    {
        painter.LoadPixels(image.getPixelsArr());
        painter.CreateCircleUsingRadius(event->pos(), radius);
        setPixmap(image.toQPixmap(1));
    }

    else if (square)
    {
        QPoint point(event->pos().x() + a, event->pos().y() + a);
        painter.LoadPixels(image.getPixelsArr());
        painter.CreateSquare(event->pos(), point);
        setPixmap(image.toQPixmap(1));
    }

    else if (rotate)
        RotateLeftPoint = event->pos();

    else if (event->button() == Qt::LeftButton)
    {
        access = true;
        startPoint = event->globalPos();
    }
}

void DragableFrame::mouseMoveEvent(QMouseEvent *event)
{
    if (access) {
        move(pos() + event -> globalPos() - startPoint);
        startPoint = event->globalPos();
    }
}

void DragableFrame::mouseReleaseEvent(QMouseEvent *event)
{
    if (rotate)
    {
        Rotate rt;
        rt.LoadSize(image.getWidth(), image.getHeight());
        rt.LoadPixels(image.getPixelsArr());

        switch (degree)
        {
            case 1:
                rt.RotateOn90Part(RotateLeftPoint, event->pos());
                break;
```

```
            case 2:
                rt.RotateOn180Part(RotateLeftPoint, event->pos());
                break;
            case 3:
                rt.RotateOn270Part(RotateLeftPoint, event->pos());
                break;
        }
        setPixmap(image.toQPixmap(1));
    }

    if (event->button() == Qt::LeftButton)
    {
        access = false;
    }
}

void DragableFrame::activateDoingRotate(int new_degree)
{
    degree = new_degree;
    rotate = true;
    square = false;
    circle = false;
}

Image* DragableFrame::getImg()
{
    return &image;
}

Painter* DragableFrame::getPnt()
{
    return &painter;
}

void DragableFrame::activateDoingSquare(int line_size)
{
    a = line_size;
    square = true;
    circle = false;
    rotate = false;
}

void DragableFrame::activateDoingCircle(int value)
{
    radius = value;
    circle = true;
    square = false;
    rotate = false;
}
```

3. Окна для подзадач.

Для выбора опций в подзадачах реализованы дополнительные окна в приложении. Каждое окно – отдельный класс, наследованный от QDialog. В общем случае, каждое окно содержит слои, которые позволяют расположить виджеты в нужном порядке, а также кнопки “Ok” и “Cancel”. А также методы возвращающие значения введенных данных. В конструкторе данного класса был реализован оператор выбора switch ответственный за создание нужного диалогового окна.

```
#include "dialog.h"

Dialog::Dialog(int temp)
{
    layout = new QGridLayout;
    pcmdOk = new QPushButton("&Ok");
    pcmdCancel = new QPushButton("&Cancel");
    connect(pcmdOk, SIGNAL(clicked()), SLOT(accept()));
    connect(pcmdCancel, SIGNAL(clicked()), SLOT(reject()));

    switch (temp) {
        case 1:
            SquareDialog();
            break;
        case 2:
            RgbDialog();
            break;
        case 3:
            CircleAndRadiusDialog();
            break;
        case 4:
            CircleAndSquareDialog();
            break;
        case 5:
            ThicknessDialog();
            break;
        case 6:
            RotateDialog(1);
            break;
        case 7:
            RotateDialog(0);
            break;
    }

    setLayout(layout);
}

Dialog::~Dialog()
{
    delete layout;
    delete pcmdOk;
    delete pcmdCancel;
}

void Dialog::SquareDialog()
{
    line_size = new QLineEdit;
    QLabel* side_size = new QLabel("Side size: ");

    QLabel* info_about_point = new QLabel("The coordinates of the upper left corner\n"
                                          "are determined by the first click on the
image.\n"
                                          "Other characteristics you can change\nby
clicking on the Line format button");

    layout->addWidget(side_size, 0, 0);
    layout->addWidget(line_size, 0, 1);
    layout->addWidget(info_about_point, 2, 0);
    layout->addWidget(pcmdOk, 3, 0);
    layout->addWidget(pcmdCancel, 3, 1);
    setLayout(layout);
}

void Dialog::ThicknessDialog()
{
```

```
    thick = new QLineEdit;
    QLabel* set_thickness = new QLabel("Set thickness: ");

    layout->addWidget(set_thickness, 0, 0);
    layout->addWidget(thick, 0, 1);
    layout->addWidget(pcmdOk, 2, 0);
    layout->addWidget(pcmdCancel, 2, 1);
}

void Dialog::RgbDialog()
{
      red_box = new QComboBox;
      green_box = new QComboBox;
      blue_box = new QComboBox;
      QStringList list;
      list << "Select number" << "0" << "255";

      red_box->addItems(list);
      green_box->addItems(list);
      blue_box->addItems(list);

      QLabel* red_cmp = new QLabel("Select the value of the red component: ");
      QLabel* green_cmp = new QLabel("Select the value of the green component: ");
      QLabel* blue_cmp = new QLabel("Select the value of the blut component: ");

      layout->addWidget(red_cmp, 0, 0);
      layout->addWidget(red_box, 0, 1);
      layout->addWidget(green_cmp, 1, 0);
      layout->addWidget(green_box, 1, 1);
      layout->addWidget(blue_cmp, 2, 0);
      layout->addWidget(blue_box, 2, 1);
      layout->addWidget(pcmdOk, 3, 0);
      layout->addWidget(pcmdCancel, 3, 1);
}

void Dialog::CircleAndRadiusDialog()
{
    QLabel* enter_radius = new QLabel("Enter radius: ");
    QLabel* info_label = new QLabel("Center coordinates are determined\nby clicking on
the picture");
    radius = new QLineEdit;

    layout->addWidget(enter_radius, 0, 0);
    layout->addWidget(radius, 0, 1);
    layout->addWidget(info_label, 1, 0);
    layout->addWidget(pcmdOk, 3, 0);
    layout->addWidget(pcmdCancel, 3, 1);
}

void Dialog::CircleAndSquareDialog()
{
     QLabel* up_crd = new QLabel("Enter the coordinates of the upper left corner: ");
     QLabel* down_crd = new QLabel("Enter the coordinates of the lower right corner: ");

     x_left_coord = new QLineEdit;
     y_left_coord = new QLineEdit;
     x_right_coord = new QLineEdit;
     y_right_coord = new QLineEdit;

     layout->addWidget(up_crd, 0, 0);
     layout->addWidget(x_left_coord, 0, 1);
     layout->addWidget(y_left_coord, 0, 2);
     layout->addWidget(down_crd, 1, 0);
     layout->addWidget(x_right_coord, 1, 1);
     layout->addWidget(y_right_coord, 1, 2);
```

```
     layout->addWidget(pcmdOk, 2, 1);
     layout->addWidget(pcmdCancel, 2, 2);
}

void Dialog::RotateDialog(int f)
{
    QLabel* select_degree = new QLabel("Select the degree of rotation:");
    degree = new QComboBox;
    degree->addItem("Select number");
    degree->addItem("90");
    degree->addItem("180");
    degree->addItem("270");

    layout->addWidget(select_degree, 0, 0);
    layout->addWidget(degree, 0, 1);
    layout->addWidget(pcmdOk, 2, 0);
    layout->addWidget(pcmdCancel, 2, 1);
    if (!f)
    {
        QLabel* label = new QLabel("Coordinates of upper left corner\n"
                                   "are determined by press on the picture\n"
                                   "Coordinates of bottom right corner\n"
                                   "are determined by mouse release");
        layout->addWidget(label, 1, 0);
    }
}

int Dialog::getSquareLineSize()
{
    return line_size->text().toInt();
}

int Dialog::getLineThickness()
{
    return thick->text().toInt();
}

int Dialog::getRgbNumber1()
{
    return red_box->currentIndex();
}

int Dialog::getRgbNumber2()
{
    return green_box->currentIndex();
}

int Dialog::getRgbNumber3()
{
    return blue_box->currentIndex();
}

int Dialog::getLeftXCoordinate()
{
    return x_left_coord->text().toInt();
}

int Dialog::getLeftYCoordinate()
{
    return y_left_coord->text().toInt();
}

int Dialog::getRightXCoordinate()
{
    return x_right_coord->text().toInt();
```

```
}

int Dialog::getRightYCoordinate()
{
    return y_right_coord->text().toInt();
}

int Dialog::getCircleRadius()
{
    return radius->text().toInt();
}

int Dialog::getDegree()
{
    return degree->currentIndex();
}
```

4. Rotate.

Данный класс предназначен для решения подзадачи связанной с поворотами изображения. В нем было реализовано 2 метода Load…() реализованных для загрузки в данный класс корректной версии изображения и его размеров. Далее в зависимости от поставленных пользователем задач вызывается одна из 6 функций. Первые 3 отвечают за поворот части изображения на 90/180/270 градусов. Был реализован поворот части изображения на 90 и 180 градусов, а для реализации поворота на 270 градусов были последовательно применены методы поворота 180 и 90 градусов. Реализация поворота на 90 и 180 градусов заключается в сохранении выделенного пользователем прямоугольника в промежуточный буффер с дальнейшим закрышиванием его в белый цвет. Далее были подсчитаны новые координаты левого верхнего и правого нижнего угла. Далее была осуществлена вставка данного прямоугольника в исходное изображение. Для реализации поворота всего изображения была реализована лишь одна функция поворота 90 градусов. Поворот на 180 и 270 градусов осуществляется при помощи данной функции. Суть данной функции аналогична сути функции поворота части изображения на 90 градусов.

```
#include "rotate.h"

Rotate::Rotate()
{
}

void Rotate::LoadPixels(PIXEL** new_pixels)
{
    pixels = new_pixels;
}

void Rotate::LoadSize(int new_width, int new_height)
{
    width = new_width;
    height = new_height;
}

void Rotate::RotateOn180Full()
{
    RotateOn90Full();
    RotateOn90Full();
}
```

```
void Rotate::RotateOn90Full()
{
     PIXEL** PixelsAfterRotate = new PIXEL*[width];
     for (int i = 0; i < width; i++) PixelsAfterRotate[i] = new PIXEL[height];

     for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
         PixelsAfterRotate[j][i] = pixels[height - 1 - i][j];

     long tmp = width;
     width = height;
     height = tmp;

     for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
         pixels[i][j] = PixelsAfterRotate[i][j];
}

void Rotate::RotateOn270Full()
{
    RotateOn90Full();
    RotateOn90Full();
    RotateOn90Full();
}

void Rotate::RotateOn90Part(QPoint start_point, QPoint current_point)
{
    int w = current_point.x() - start_point.x(),
        h = current_point.y() - start_point.y(),
        x1 = start_point.x(),
        y1 = start_point.y(),
        x2 = current_point.x(),
        y2 = current_point.y();

    if (x1 > width || x2 > width || y1 > height || y2 > height || x1 > x2 || y1 > y2)
        return;
    int max = h;
    if (h < w) max = w;

    PIXEL** buf_tmp = new PIXEL* [max];
    for (int i = 0; i <= max; i++) buf_tmp[i] = new PIXEL[max];

    for (int i = y1; i <= y2; i++) for (int j = x1; j <= x2; j++){
        buf_tmp[i - y1][j - x1] = pixels[i][j];
        pixels[i][j] = {255, 255, 255};
    }
    int len, x_left, y_left, x_right, y_right;
    if (h < w)
    {
        len = (w - h) / 2;
        x_left = x1 + len + ((w - h) % 2);
        y_left = y1 - len - ((w - h) % 2);
        x_right = x2 - len;
        y_right = y2 + len;
    }
    else
    {
        len = (h - w) / 2;
        x_left = x1 - len - ((w - h) % 2);
        y_left = y1 + len + ((w - h) % 2);
        x_right = x2 + len;
        y_right = y2 - len;
    }

    for (int i = y_left; i < y_right; i++) for (int j = x_left; j < x_right; j++)
    {
```

```
        int n = h - (j - x_left);
        int m = i - y_left;
        if (i >= 0 && j >= 0 && i < height && j < width &&
            n >= 0 && n < h && m >= 0 && m < w)
            pixels[i][j] = buf_tmp[n][m];
    }
}

void Rotate::RotateOn180Part(QPoint start_point, QPoint current_point)
{
    int w = current_point.x() - start_point.x(),
        h = current_point.y() - start_point.y(),
        x1 = start_point.x(),
        y1 = start_point.y(),
        x2 = current_point.x(),
        y2 = current_point.y();

    int max = h;
    if (h < w) max = w;

    PIXEL** buf_tmp = new PIXEL* [max];
    for (int i = 0; i <= max; i++) buf_tmp[i] = new PIXEL[max];

    for (int i = y1; i <= y2; i++) for (int j = x1; j <= x2; j++)
        buf_tmp[i - y1][j - x1] = pixels[i][j];

    for (int i = y1; i <= y2; i++) for (int j = x1; j <= x2; j++)
        if ((i >= 0) && (j >= 0) && (i < height) && (j < width))
            pixels[i][j] = buf_tmp[h - (i - y1)][w - (j - x1)];
}

void Rotate::RotateOn270Part(QPoint start_point, QPoint current_point)
{
    RotateOn180Part(start_point, current_point);
    RotateOn90Part(start_point, current_point);
}
```

5. Image.

Image – класс, позволяющий хранить файлы формата .bmp 3, 4 и 5 версий. Для хранения реализованы структуры: 2 заголовка и PIXEL структура, содержащая в себе 3 поля отвечающих за оттенок каждой из компонент. Для отключения выравнивания структур использована утилита #pragma pack(push, 1) и #pragma pack(pop).

```
#pragma pack(push, 1)
typedef struct tagBITMAPFILEHEADER{
    unsigned short  bfType;
    unsigned long   bfSize;
    unsigned short  bfReserved1;
    unsigned short  bfReserved2;
    unsigned long   bfOffBits;
} BITMAPFILEHEADER;
#pragma pack(pop)


typedef struct tagBITMAPINFOHEADER{
    unsigned int    biSize;
    long        biWidth;
    long        biHeight;
    unsigned short  biPlanes;
    unsigned short  biBitCount;
```

```
    unsigned int    biCompression;
    unsigned int    biClrUsed;
} BITMAPINFOHEADER;


struct PIXEL{
        unsigned char r;
        unsigned char g;
        unsigned char b;
};

class Image
{

public:
    Image();
    ~Image();
    int load(const char* fname);
    int save(const char* fname);
    void create();
    void SwapCoordinates();
    long int getWidth();
    long int getHeight();
    void LoadOldPixmap();
    void ShiftOldPixmaps();
    QPixmap toQPixmap(int f);
    QPixmap toOldQPixmap();
    QString info() const;
    PIXEL** getPixelsArr();

private:
    int max;
    PIXEL** pixels;
    BITMAPFILEHEADER  bfh;
    BITMAPINFOHEADER  bih;
    PIXEL*** OldPixmaps;
    int PixmapIndex;
};
```

Для считывания нашей картинки была реализована функция load. В данной функции сначала считываются 2 заголовка нашего файла. Далее выполняется проверка на корректность файла. Если файл прошел проверку, то выполняется его считывание. Создается массив пикселей, в который при помощи вложенного цикла заносятся оттенки каждого из 3-х основных цветов. Если все прошло успешно, то наше функция возвращает 1 иначе -1 или 0.

```
int Image::load(const char* fname)
{
    FILE *f;
    f = fopen(fname,"r+b");
    fread(&bfh, sizeof(BITMAPFILEHEADER), 1, f);
    fread(&bih, sizeof(BITMAPINFOHEADER), 1, f);

    if (bfh.bfReserved1 != 0 ||
            (bih.biSize != 40 && bih.biSize != 108 && bih.biSize != 124) ||
            bih.biWidth < 1 || bih.biWidth > 10000 ||
            bih.biHeight < 1 || bih.biHeight > 10000 ||
            bih.biCompression != 0 ||
            bih.biBitCount != 24)
        return -1;

    max = bih.biHeight;
```

```
    if (max < bih.biWidth) max = bih.biWidth;

    OldPixmaps[PixmapIndex] = new PIXEL*[bih.biHeight];
    for (int i = 0; i < bih.biHeight; i++)
        OldPixmaps[PixmapIndex][i] = new PIXEL[bih.biWidth];

    pixels = new PIXEL*[max];
    for (int i = 0; i < max; i++) pixels[i] = new PIXEL[max];

    fseek(f, static_cast<long>(bfh.bfOffBits), SEEK_SET);

    for (int i = 0; i < bih.biHeight; i += 1) for (int j = 0; j < bih.biWidth; j += 1)
    {
        fread(&pixels[bih.biHeight - 1 - i][j].b, 1, 1, f);
        fread(&pixels[bih.biHeight - 1 - i][j].g, 1, 1, f);
        fread(&pixels[bih.biHeight - 1 - i][j].r, 1, 1, f);
    }

    fclose(f);
    return 1;
  }
```

Также была реализована важная функция toQPixmap() которая преобразует наш массив в обьект класса QPixmap для удобства дальнейшего его изображения. Ее реализация заключается в добавлении каждого пикселя из нашего массива в библиотечный класс созданный для хранения изображения. А далее из него при помощи метода convertFromImage() достается необходимый нам QPixmap.

```
QPixmap Image::toQPixmap(int f)
{
    QPixmap pix;
    QImage* pict = new QImage(bih.biWidth, bih.biHeight, QImage::Format_RGB888);

    for (int i = 0; i < bih.biHeight; i++) for (int j = 0; j < bih.biWidth; j++)
    {
        QColor color(pixels[i][j].r, pixels[i][j].g, pixels[i][j].b);
        pict->setPixel(j, i, color.rgb());
    }
    pix.convertFromImage(*pict, nullptr);

    if (f)
        LoadOldPixmap();

    return pix;
  }
```

Также была реализована функция по сохранению/созданию файла. Первая из данных функция была реализована аналогично функции загрузке изображения. Для реализации второй функции потребовалось изменить поля структуры на стандартные и закрасить наше изображения белым цветом. Данная подзадача была реализована при помощи циклов.

```
  void Image::create()
  {
      memset(&bih, 0, sizeof(bih));
      memset(&bfh, 0, sizeof(bfh));
```

```
    bih.biWidth = 1280;
    bih.biHeight = 720;
    bih.biSize = sizeof(bih);
    bih.biPlanes = 1;
    bih.biBitCount = 24;
    bfh.bfOffBits = 14 + sizeof(bih);
    bfh.bfType = 0x4d42;
    bfh.bfSize = static_cast<DWORD>(54 + bih.biHeight * bih.biWidth);

    for (int i = 0; i < bih.biHeight; i++) for (int j = 0; j < bih.biWidth; j++)
        pixels[i][j] = {255, 255, 255};
     }
```

Также была реализована функция по получения информации о файле. При помощи данных, хранящихся в наших обьектах структур можно получить всю необходимую информацию и записать ее в обьект класса QString.

```
QString Image::info() const
{
    QString information;
    QString n = QString::number(static_cast<double>(bfh.bfSize) / 1048576, 'g', 3);
    information.sprintf("File format: .bmp\nFile resolution: %ld x %ld\n",
bih.biWidth, bih.biHeight);
    information += "Size on disk: " + n + " Mb\n";
    information += "Bit count: " + QString::number(bih.biBitCount);
    return information;
     }
```

6. Painter

Этот класс отвечает за решение 3-х поставленных подзадач, а также функции ответственные за установление цвета линий/заливки и толщины линии.

7. Square

Для рисования квадрата с диагоналями была создана функция CreateSquare(QPoint start_point, QPoint current_point). Start_point – переменная хранящая в себе координаты левого верхнего угла. Current_point - переменная хранящая в себе координаты правого нижнего угла. При помощи переменных k, b и k2, b2 были получены уравнения диагоналей, задаваемые уравнениями i = kj + b. Далее был осуществлен проход по нашему массиву и закрашены те пиксели, которые лежат на диагоналях и на сторонах квадрата. Также была реализована функция drawPixel(int x, int y, QColor cur_color), которая окрашивала нужный пиксель в переданный цвет.

```
void Painter::CreateSquare(QPoint start_point, QPoint current_point)
{
    int x1 = start_point.x(),
        y1 = start_point.y(),
        x2 = current_point.x(),
        y2 = current_point.y();
    double k = static_cast<double>(y2 - y1) / static_cast<double>(x2 - x1);
    double b = static_cast<double>(y1) - k * x1;
    double k2 = -k;
    double b2 =  static_cast<double>(y1) - k2 * x2;

    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
```

```
    {
        if (fill && i > y1 && i < y2
                && j > x1 && j < x2)
            drawPixel(i, j, fill_color);

        if ((abs(i - j * k - b) <= thickness * 2 || abs(i - j * k2 - b2) <= thickness *
2)
                && i >= y1 && i <= y2
                && j >= x1 && j <= x2)
            drawPixel(i, j, CurrentLineColor);

        if ((abs(i - y1) <= thickness || abs(i - y2) <= thickness) && j >= x1 && j <=
x2)
            drawPixel(i, j, CurrentLineColor);

        if ((abs(j - x1) <= thickness || abs(j - x2) <= thickness) &&
            i >= y1 - thickness && i <= y2 + thickness)
            drawPixel(i, j, CurrentLineColor);
    }
   }
void Painter::drawPixel(int x, int y, QColor cur_color)
{
    pixels[x][y].r = static_cast<unsigned char>(cur_color.red());
    pixels[x][y].g = static_cast<unsigned char>(cur_color.green());
   pixels[x][y].b = static_cast<unsigned char>(cur_color.blue());}
```

8. Circle

Для рисования окружности было создано 2 функции **CreateCircleUsingRadius**(QPoint center, int radius) и **CreateCircleUsingSquare**(QPoint left, QPoint right). Первая отвечает за рисование окружности, используя координаты центра и радиус, ее реализация заключается в проходе по массиву пикселей и окрашивании тех пикселей, которые лежат на окружности, это было определено при помощи уравнения окружности и некоторой точности. Вторая функция отвечает за рисование окружности вписанной в квадрат, задаваемый координатами левого верхнего угла и координатами правого нижнего угла. При помощи математематических завистей были найдены координаты центра окружности и ее радиус, а далее применена первая функция. Для окрашивания использовалась вышеописанная функция drawPixel().

```
void Painter::CreateCircleUsingRadius(QPoint center, int radius)
{
    int x = center.x(),
        y = center.y();
    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
    {
        if (fill && (j - x)*(j - x) + (i - y)*(i - y) - radius * radius < 0)
            drawPixel(i, j, fill_color);

        if (abs((j - x)*(j - x) + (i - y)*(i - y) - radius * radius) <= radius * 3 *
thickness)
            drawPixel(i, j, CurrentLineColor);
    }
}

void Painter::CreateCircleUsingSquare(QPoint left, QPoint right)
{
    int a = right.x() - left.x();
```

```
    int radius = a / 2;
    QPoint center(left.x() + radius, left.y() + radius);
    CreateCircleUsingRadius(center, radius);
  }
```

Функции отвечащие за изменение цвета линий/заливки, если она активирована, а также толщины линий.

```
void Painter::UpdateColor(QColor new_color)
{
    CurrentLineColor = new_color;
}

void Painter::UpdateFillColor(QColor new_color)
{
    fill = true;
    fill_color = new_color;
}

void Painter::UpdateThickness(int new_thickness)
{
    thickness = new_thickness;
}

void Painter::disconnectFill()
{
    fill = false;}
```

9. Rgb

Для решения подзадачи связанной с rgb-компонентом. Была реализована функция CreateRgb(), которая принимает номер цвета, значение которого нужно изменить, и значение, в которое его нужно изменить. Далее был осуществлен проход по массиву пикселей с целью изменения значения нужной компоненты в необходимое значение.

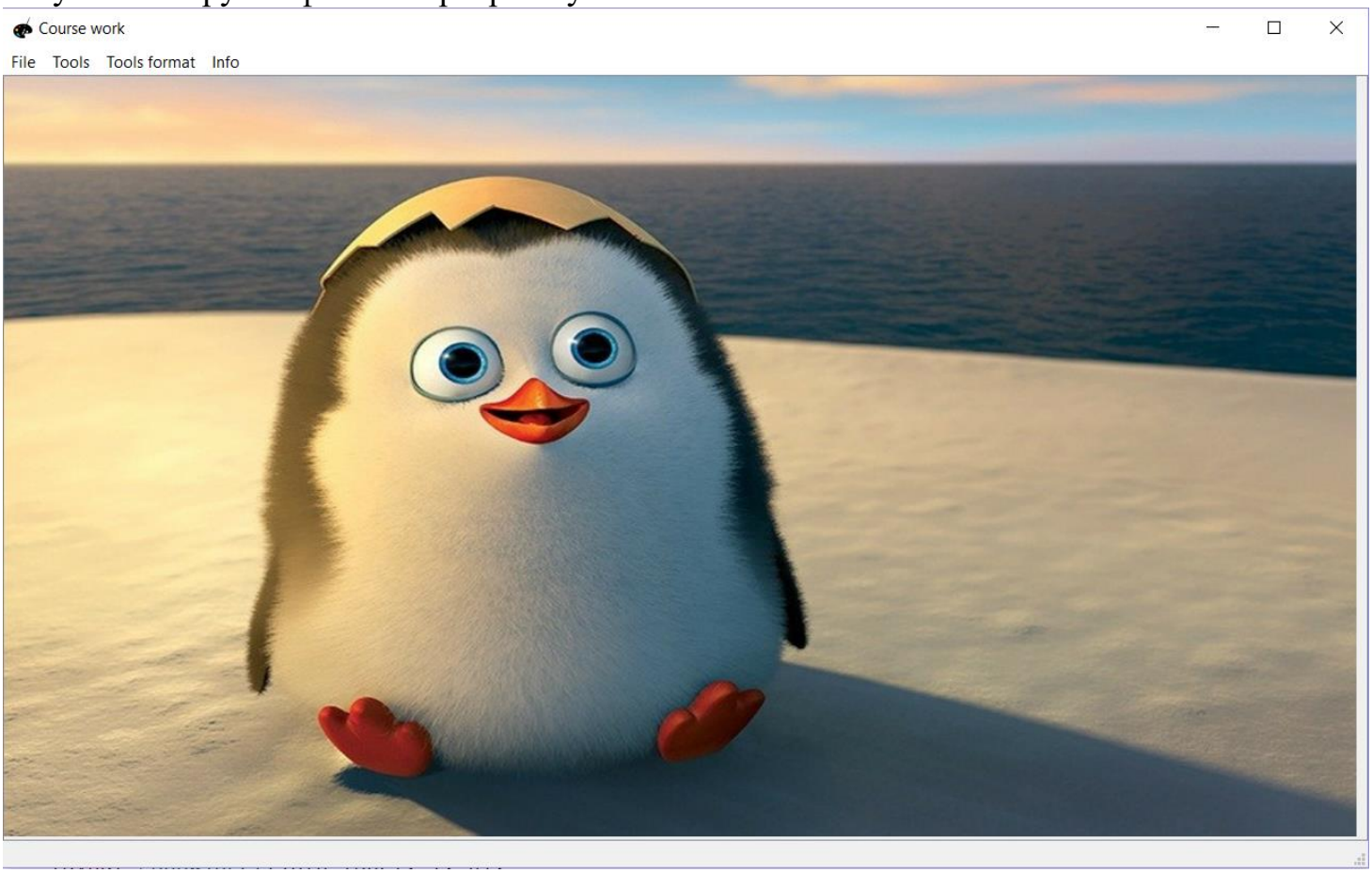
```
void Painter::CreateRgb(int n_color, int n)
{
    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
    {
        if (!n_color)
            pixels[i][j].r = static_cast<unsigned char>(n);
        else if (n_color == 1)
            pixels[i][j].g = static_cast<unsigned char>(n);
        else
            pixels[i][j].b = static_cast<unsigned char>(n);
    }


  }
```

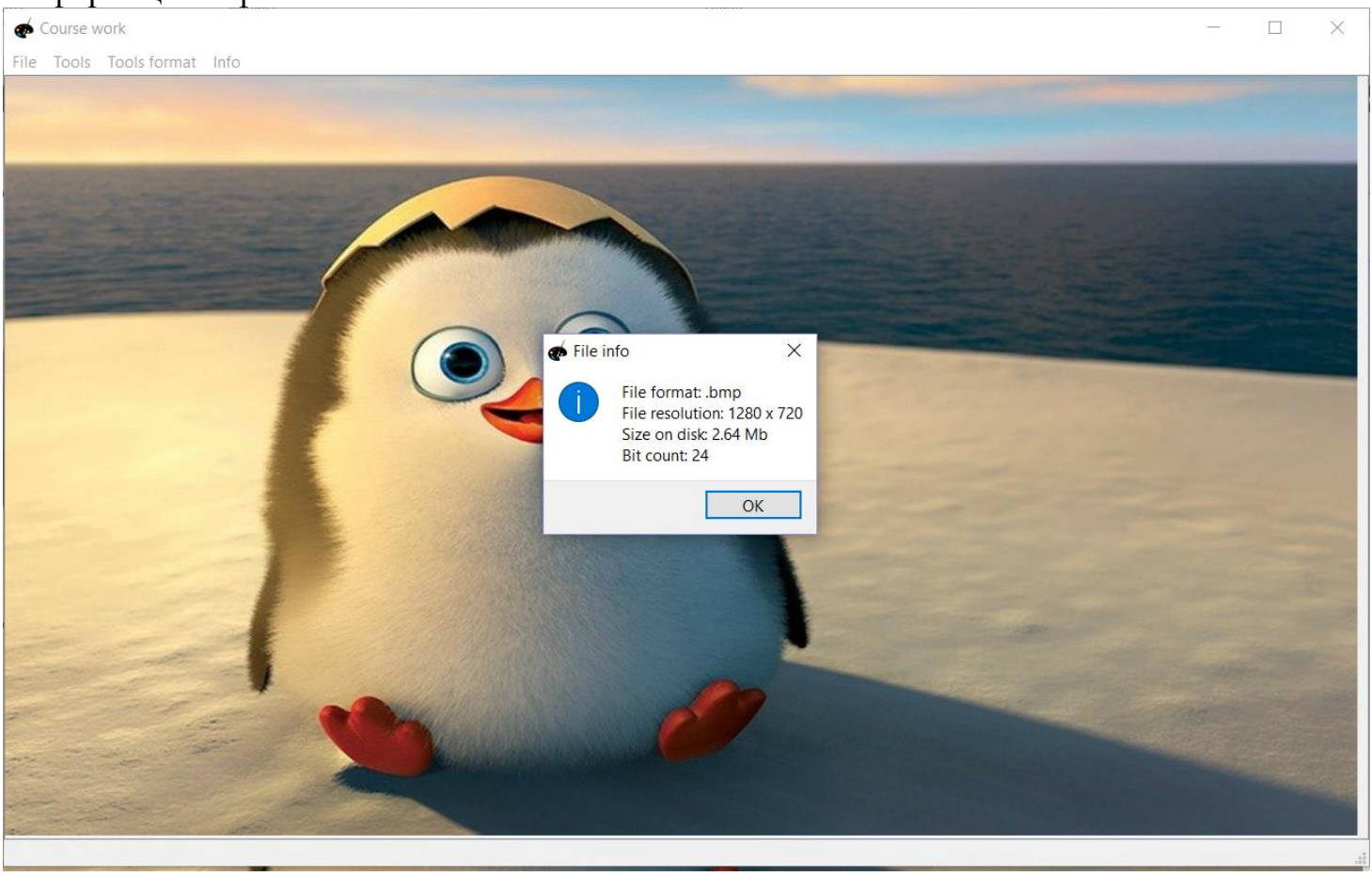
Тестирование программы.

Исходный файл:

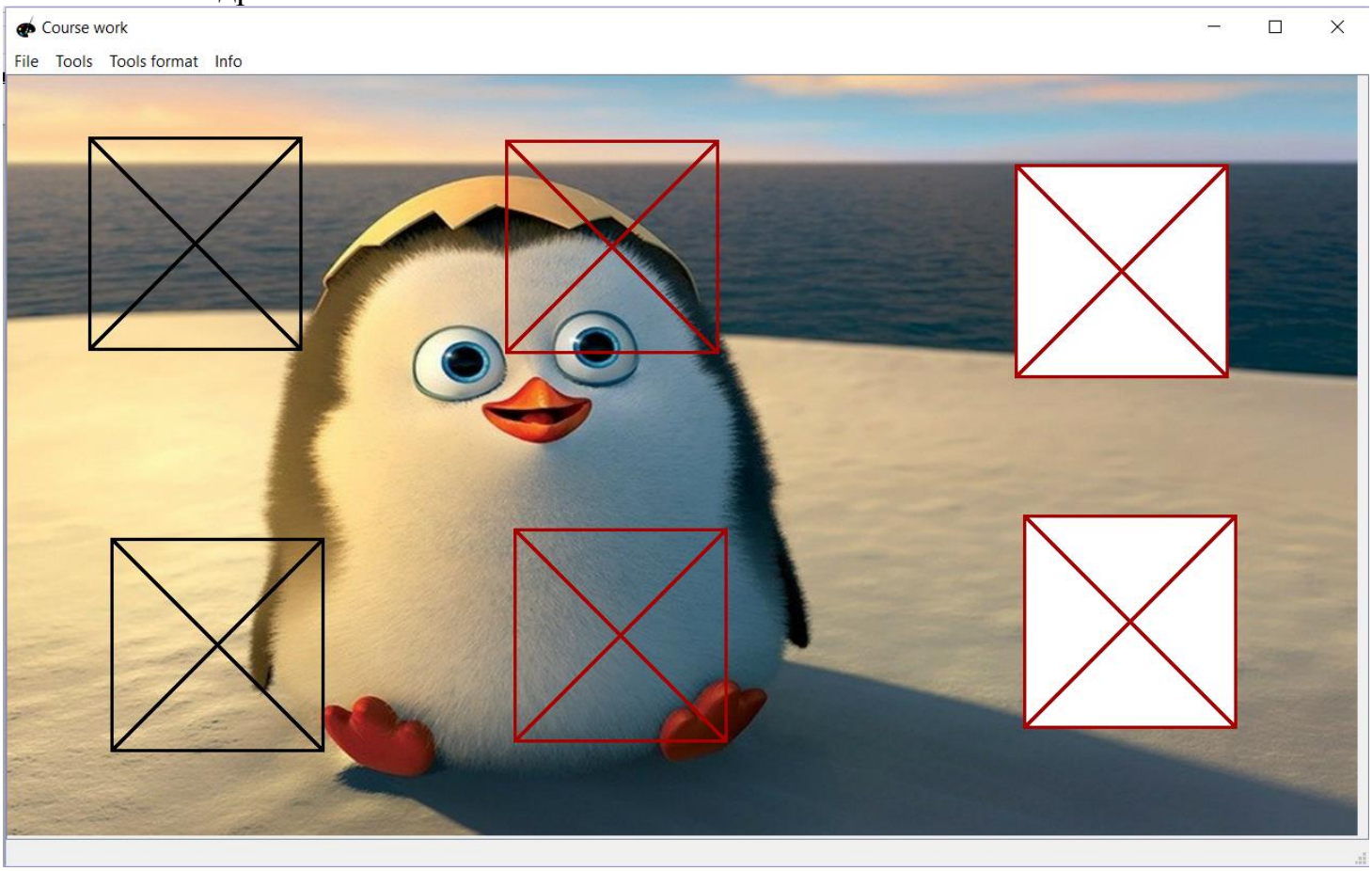
Результат загрузки файла в программу:



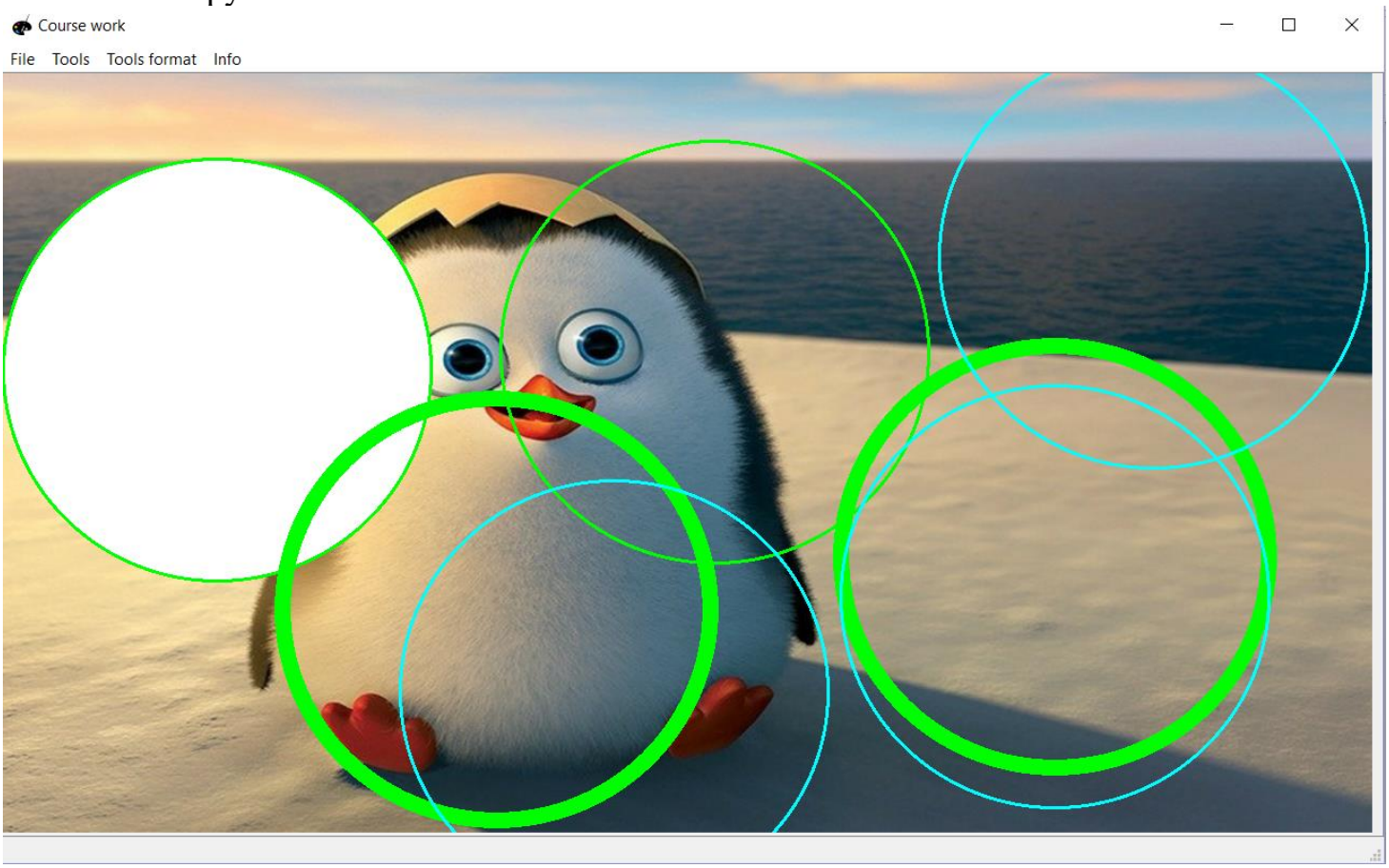
Информация о файле:



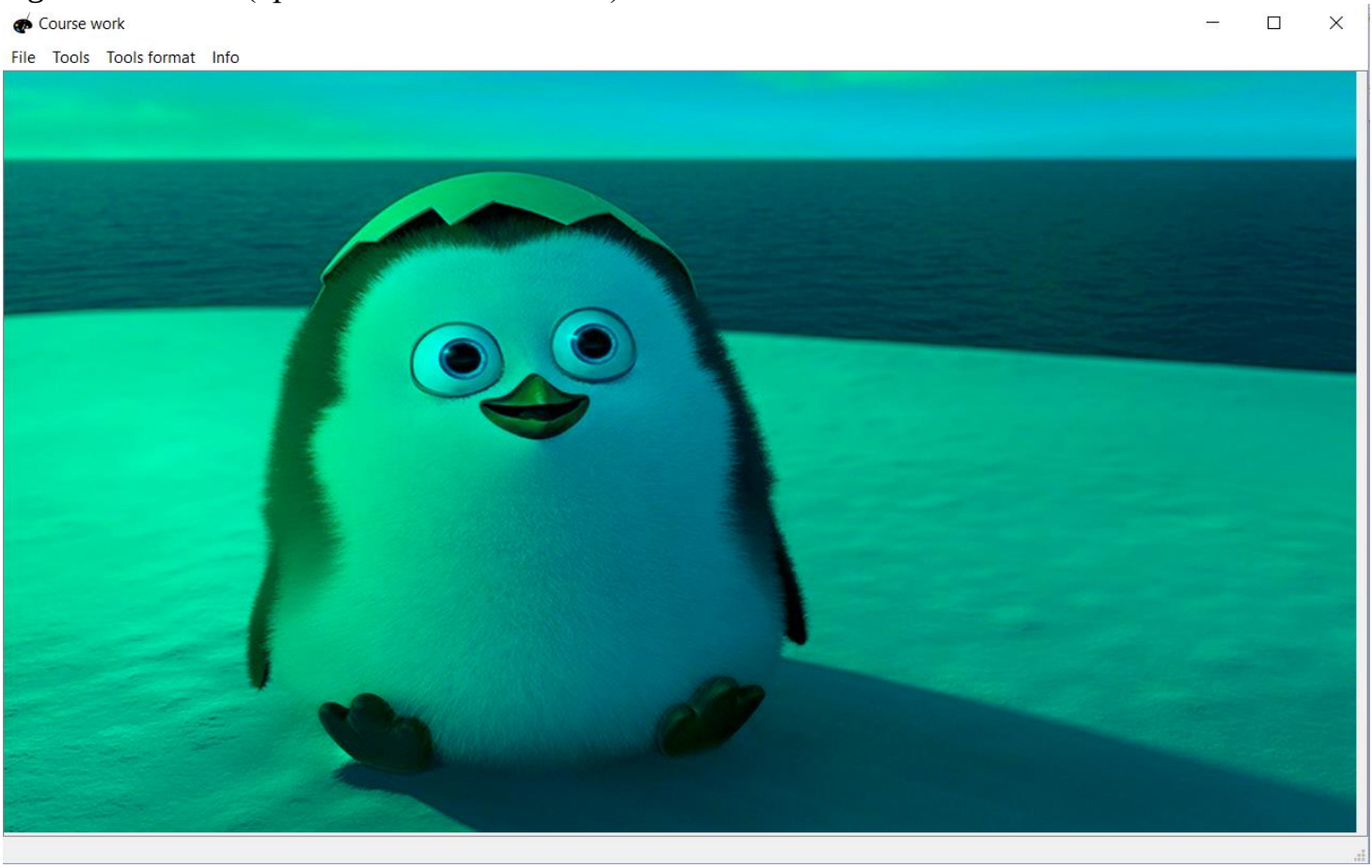
Рисование квадрата:



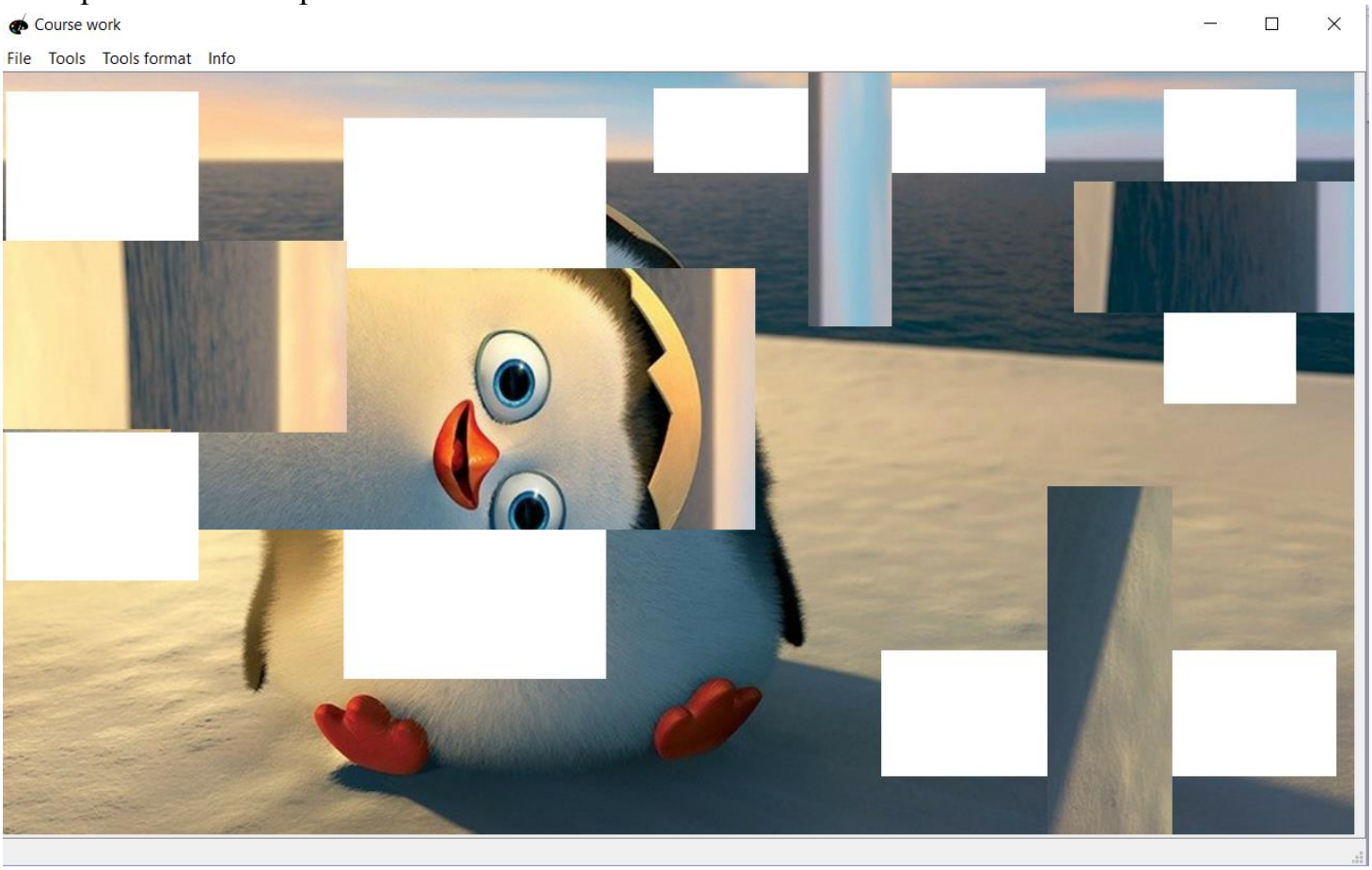
Рисование окружности:



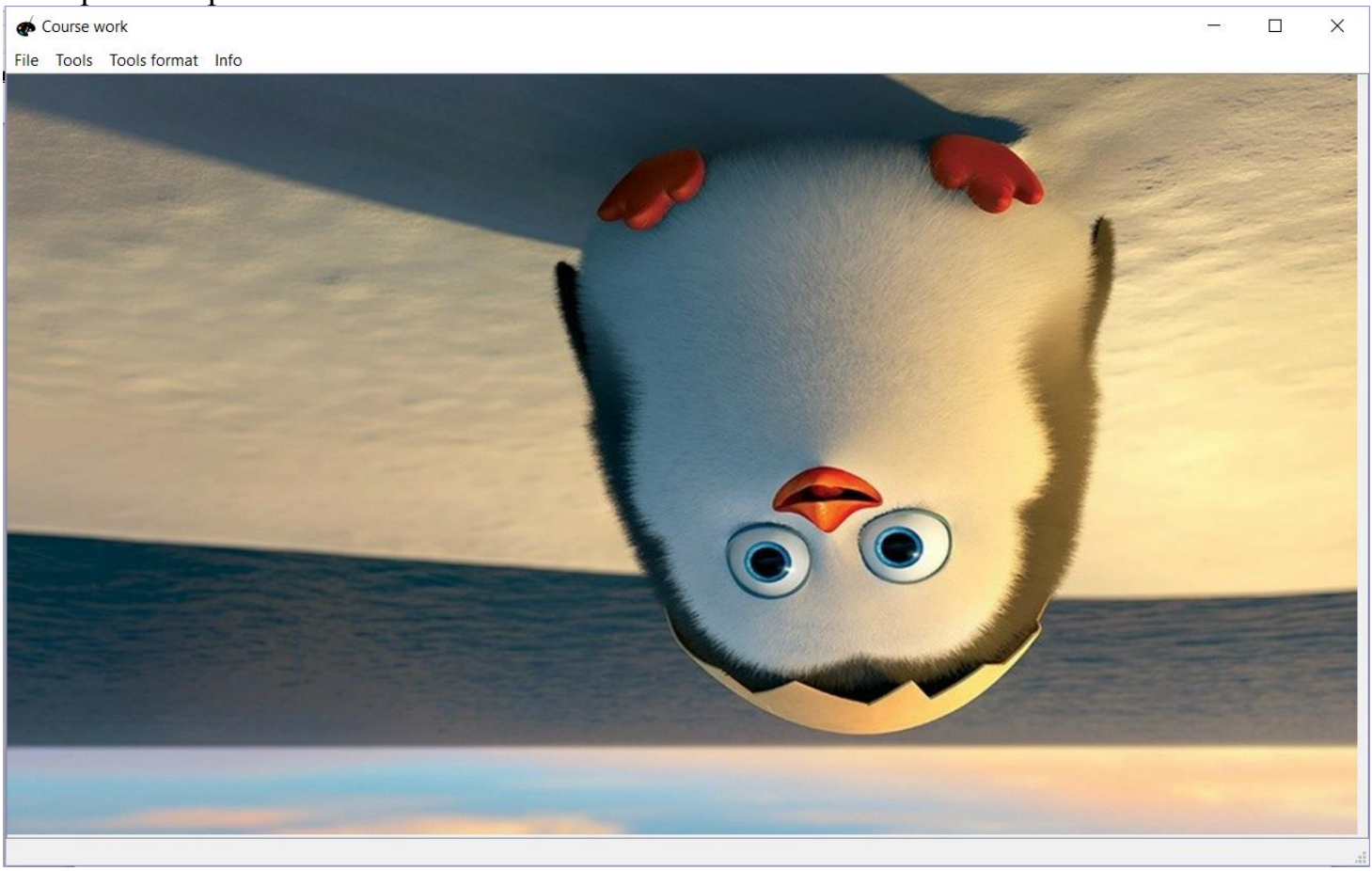
Rgb-компонент (красная компонента в 0):



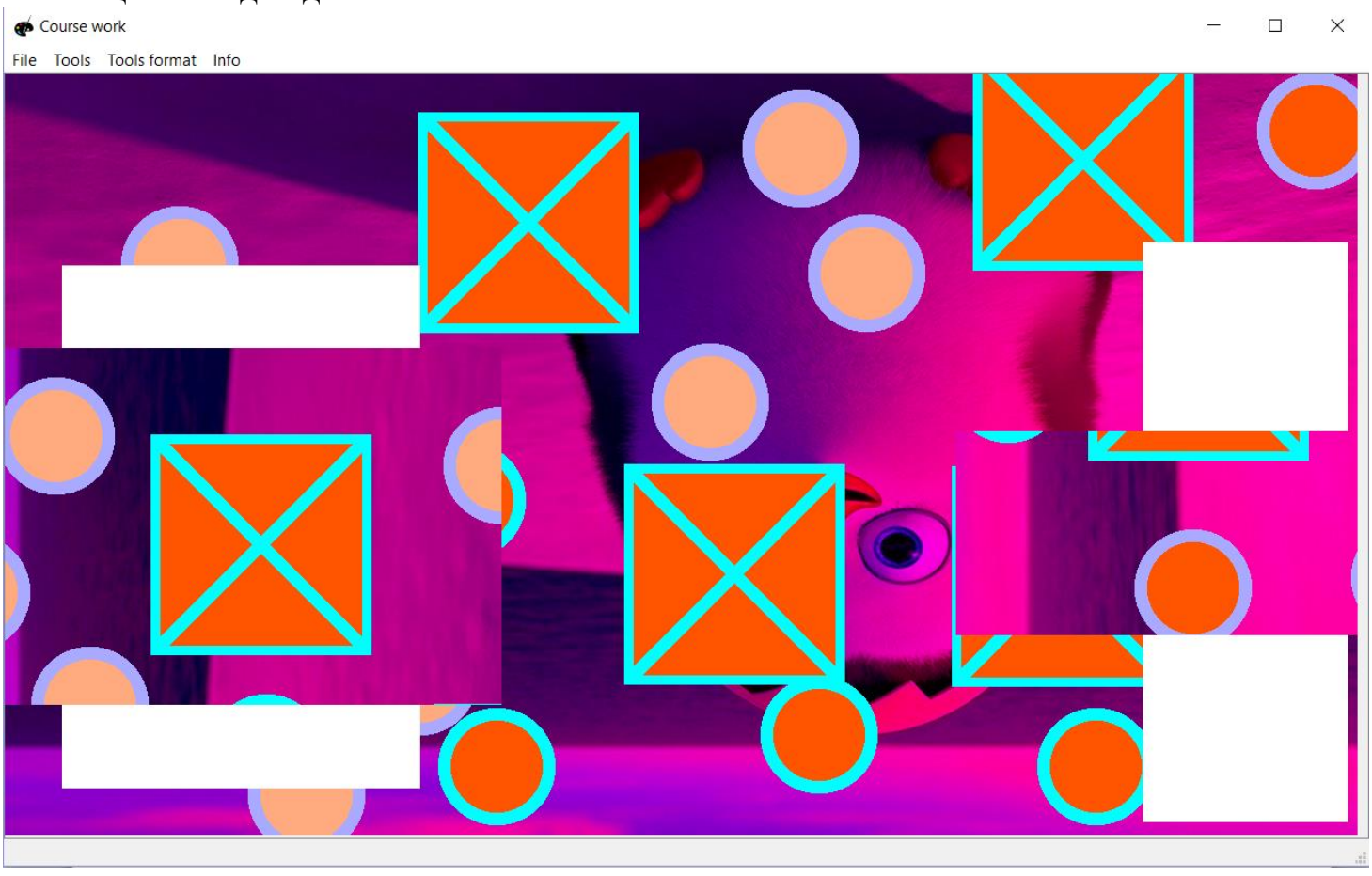
Поворот части изображения:



Поворот изображения:



Совмещение подазадач:



Заключение.

В ходе выполнения работы было создано desktop-приложение с GUI для обработки файлов-изображений в формате bmp. Для работы программы были созданы и описаны все необходимые классы и структуры. Программа была протестирована, результат работы программы соответствует заданным условиям.

Список использованных источников.

1. https://ru.wikipedia.org/wiki/
2. Документация фреймворка Qt.
3. https://www.cyberforum.ru/
4. https://prog-cpp.ru/
5. https://doc.qt.io/

Исходный код программы

- **dialog.h**

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QColor>
#include <QString>
#include <QDialog>
#include <QLineEdit>
#include <QLabel>
#include <QPushButton>
#include <QGridLayout>
#include <QColorDialog>
#include <QDialogButtonBox>
#include <QComboBox>
#include <QMessageBox>
#include <QFormLayout>
#include <QScrollBar>

class Dialog : public QDialog
{
    Q_OBJECT

public:
    Dialog(int term);
    ~Dialog();
    int getSquareLineSize();
    int getLineThickness();
    int getRgbNumber1();
    int getRgbNumber2();
    int getRgbNumber3();
    int getLeftXCoordinate();
 int getLeftYCoordinate();
    int getRightXCoordinate();
 int getRightYCoordinate();
    int getCircleRadius();
 int getDegree();
    int getRedValueOfBW();
    int getGreenValueOfBW();
    int getBlueValueOfBW();
    void SquareDialog();
    void ThicknessDialog();
    void RgbDialog();
    void CircleAndSquareDialog();
```

```
        void CircleAndRadiusDialog();
        void RotateDialog(int f);
        void RectangleAndColorDialog();

    private:
        QLineEdit* thickness;
        QLineEdit* line_size;
        QLineEdit* radius;
        QLineEdit* x_left_coord;
        QLineEdit* y_left_coord;
        QLineEdit* x_right_coord;
        QLineEdit* y_right_coord;
        QLineEdit* thick;
        QLineEdit* red_bw;
        QLineEdit* blue_bw;
        QLineEdit* green_bw;
        QPushButton* pcmdOk;
        QPushButton* pcmdCancel;
        QGridLayout* layout;
        QComboBox* red_box;
        QComboBox* green_box;
        QComboBox* blue_box;
        QComboBox* degree;
    };

#endif // DIALOG_H
```

- **image.h**

```
    #ifndef IMAGE_H
    #define IMAGE_H

    #include <QPixmap>
    #include <stdio.h>
    #include <cstdlib>
    #include <math.h>

    typedef unsigned short WORD;
    typedef unsigned long DWORD;
    typedef long LONG;


    #pragma pack(push, 1)
    typedef struct tagBITMAPFILEHEADER{
        WORD    bfType;
        DWORD   bfSize;
        WORD    bfReserved1;
        WORD    bfReserved2;
        DWORD   bfOffBits;
    } BITMAPFILEHEADER;
    #pragma pack(pop)


    typedef struct tagBITMAPINFOHEADER{
        DWORD  biSize;
        LONG   biWidth;
        LONG   biHeight;
        WORD   biPlanes;
        WORD   biBitCount;
        DWORD  biCompression;
        DWORD  biSizeImage;
        LONG   biXPelsPerMeter;
        LONG   biYPelsPerMeter;
        DWORD  biClrUsed;
        DWORD  biClrImportant;
    } BITMAPINFOHEADER;
```

```
struct PIXEL{
   unsigned char b;
        unsigned char g;
        unsigned char r;
};

class Image
{

public:
    Image();
    ~Image();
    int load(const char* fname);
    int save(const char* fname);
    void create();
    void SwapCoordinates();
    long int getWidth();
    long int getHeight();
    void LoadOldPixmap();
    void ShiftOldPixmaps();
    QPixmap toQPixmap(int f);
    QPixmap toOldQPixmap();
    QString info() const;
    PIXEL** getPixelsArr();

private:
    int max;
    PIXEL** pixels;
    BITMAPFILEHEADER  bfh;
    BITMAPINFOHEADER  bih;
    PIXEL*** OldPixmaps;
    int PixmapIndex;
};

#endif // IMAGE_H
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QFileDialog>
#include <QMessageBox>
#include <QColorDialog>
#include <QColor>
#include <QPixmap>
#include <cstring>
#include <QScrollArea>
#include "dragableframe.h"
#include "dialog.h"
#include "rotate.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
```

```
 private slots:
     void on_actionOpen_triggered();
     void on_actionCreate_triggered();
     void on_actionSave_triggered();
     void Scroll();
     void checkRgb(int n, int n_color);
     int LoadFile(QString filename);
     void CreateCircleMenu();
     void CreateRotateMenu();
     void SetIcons();
     void on_actionAbout_program_2_triggered();
     void on_actionInfo_about_File_triggered();
     void on_actionChoosing_color_triggered();
     void on_actionSquare_triggered();
     void on_actionLine_thickness_triggered();
     void on_actionFill_color_triggered();
     void on_actionRgb_triggered();
     void on_actionRadius_and_center_coordinates_triggered();
     void on_actionSquare_coordinates_triggered();
     void on_RotateFullPicture_triggered();
     void on_RotatePartOfPicture_triggered();
     void on_actionStep_back_2_triggered();

     void on_actionBW_component_triggered();

     void on_actionSmallest_rectangle_triggered();

 private:
     Ui::MainWindow *ui;
     QString filename;
     QColor chosen_color;
     DragableFrame* label;
     QScrollArea* scrollArea;
     char* good_filename;
     Rotate rt;
 };

   #endif // MAINWINDOW_H
```

- **painter.h**

```
#ifndef CIRCLEANDSQUARE_H
#define CIRCLEANDSQUARE_H

#include "image.h"

class Painter
{
public:
    Painter();
    void LoadPixels(PIXEL** new_pixels);
    void LoadSize(int new_width, int new_height);
    void CreateSquare(QPoint start_point, QPoint current_point);
    void CreateCircleUsingRadius(QPoint center, int radius);
    void CreateCircleUsingSquare(QPoint left, QPoint right);
    void UpdateColor(QColor new_color);
    void UpdateFillColor(QColor new_color);
    void UpdateThickness(int new_thickness);
    void disconnectFill();
    void drawPixel(int x, int y, QColor cur_color);
    void CreateRgb(int n_color, int n);
    void CreateBW();

private:
    PIXEL** pixels;
```

```
    int width;
    int height;

    bool fill = false;
    QColor fill_color;
    QColor CurrentLineColor = Qt::black;
    int thickness = 1;
};

#endif // CIRCLEANDSQUARE_H
```

- **rotate.h**

```
#ifndef ROTATE_H
#define ROTATE_H

#include <image.h>

class Rotate
{
public:
    Rotate();
    void LoadPixels(PIXEL** new_pixels);
    void LoadSize(int new_width, int new_height);
    void RotateOn180Full();
    void RotateOn90Full();
    void RotateOn270Full();
    void RotateOn90Part(QPoint start_point, QPoint current_point);
    void RotateOn180Part(QPoint start_point, QPoint current_point);
    void RotateOn270Part(QPoint start_point, QPoint current_point);

private:
    int width;
    int height;
    PIXEL** pixels;
};

#endif // ROTATE_H
```

- **dialog.cpp**

```
#include "dialog.h"

Dialog::Dialog(int temp)
{
    layout = new QGridLayout;
    pcmdOk = new QPushButton("&Ok");
    pcmdCancel = new QPushButton("&Cancel");
    connect(pcmdOk, SIGNAL(clicked()), SLOT(accept()));
    connect(pcmdCancel, SIGNAL(clicked()), SLOT(reject()));
    pcmdOk->setStyleSheet("background-color: green; "
                          "border-style: outset; "
                          "border-radius: 10px; "
                          "font: bold 14px; "
                          "min-width: 10em; "
                          "padding: 6px;");
    pcmdCancel->setStyleSheet("background-color: red; "
                              "border-style: outset; "
                              "border-radius: 10px; "
                              "font: bold 14px; "
                              "min-width: 10em; "
                              "padding: 6px;");

    switch (temp) {
        case 1:
```

```
                SquareDialog();
                break;
            case 2:
              RgbDialog();
              break;
          case 3:
              CircleAndRadiusDialog();
              break;
          case 4:
              CircleAndSquareDialog();
              break;
          case 5:
              ThicknessDialog();
              break;
          case 6:
              RotateDialog(1);
              break;
          case 7:
              RotateDialog(0);
              break;
          case 8:
              RectangleAndColorDialog();
              break;
      }

      setLayout(layout);
}

Dialog::~Dialog()
{
      delete layout;
      delete pcmdOk;
      delete pcmdCancel;
}

void Dialog::SquareDialog()
{
      line_size = new QLineEdit;
      QLabel* side_size = new QLabel("Side size: ");

      QLabel* info_about_point = new QLabel("The coordinates of the upper left corner\n"
                                            "are determined by the first click on the
image.\n"
                                            "Other characteristics you can change\nby
clicking on the Line format button");
      info_about_point->setStyleSheet("color: red");

      layout->addWidget(side_size, 0, 0);
      layout->addWidget(line_size, 0, 1);
      layout->addWidget(info_about_point, 2, 0);
      layout->addWidget(pcmdOk, 3, 0);
      layout->addWidget(pcmdCancel, 3, 1);
}

void Dialog::ThicknessDialog()
{
      thick = new QLineEdit;
      QLabel* set_thickness = new QLabel("Set thickness: ");

      layout->addWidget(set_thickness, 0, 0);
      layout->addWidget(thick, 0, 1);
      layout->addWidget(pcmdOk, 2, 0);
      layout->addWidget(pcmdCancel, 2, 1);
}

void Dialog::RgbDialog()
```

```
{
      red_box = new QComboBox;
      green_box = new QComboBox;
      blue_box = new QComboBox;
      QStringList list;
      list << "Select number" << "0" << "255";

      red_box->addItems(list);
      green_box->addItems(list);
      blue_box->addItems(list);

      QLabel* red_cmp = new QLabel("Select the value of the red component: ");
      QLabel* green_cmp = new QLabel("Select the value of the green component: ");
      QLabel* blue_cmp = new QLabel("Select the value of the blut component: ");

      layout->addWidget(red_cmp, 0, 0);
      layout->addWidget(red_box, 0, 1);
      layout->addWidget(green_cmp, 1, 0);
      layout->addWidget(green_box, 1, 1);
      layout->addWidget(blue_cmp, 2, 0);
      layout->addWidget(blue_box, 2, 1);
      layout->addWidget(pcmdOk, 3, 0);
      layout->addWidget(pcmdCancel, 3, 1);
}

void Dialog::CircleAndRadiusDialog()
{
    QLabel* enter_radius = new QLabel("Enter radius: ");
    QLabel* info_label = new QLabel("Center coordinates are determined\nby clicking on
the picture");
    info_label->setStyleSheet("color: red;");
    radius = new QLineEdit;

    layout->addWidget(enter_radius, 0, 0);
    layout->addWidget(radius, 0, 1);
    layout->addWidget(info_label, 1, 0);
    layout->addWidget(pcmdOk, 3, 0);
    layout->addWidget(pcmdCancel, 3, 1);
}

void Dialog::CircleAndSquareDialog()
{
     QLabel* up_crd = new QLabel("Enter the coordinates of the upper left corner: ");
     QLabel* down_crd = new QLabel("Enter the coordinates of the lower right corner: ");

     x_left_coord = new QLineEdit;
     y_left_coord = new QLineEdit;
     x_right_coord = new QLineEdit;
     y_right_coord = new QLineEdit;

     layout->addWidget(up_crd, 0, 0);
     layout->addWidget(x_left_coord, 0, 1);
     layout->addWidget(y_left_coord, 0, 2);
     layout->addWidget(down_crd, 1, 0);
     layout->addWidget(x_right_coord, 1, 1);
     layout->addWidget(y_right_coord, 1, 2);
     layout->addWidget(pcmdOk, 2, 1);
     layout->addWidget(pcmdCancel, 2, 2);
}

void Dialog::RotateDialog(int f)
{
    QLabel* select_degree = new QLabel("Select the degree of rotation:");
    degree = new QComboBox;
    degree->addItem("Select number");
    degree->addItem("90");
```

```
    degree->addItem("180");
    degree->addItem("270");

    layout->addWidget(select_degree, 0, 0);
    layout->addWidget(degree, 0, 1);
    layout->addWidget(pcmdOk, 2, 0);
    layout->addWidget(pcmdCancel, 2, 1);
    if (!f)
    {
        QLabel* label = new QLabel("Coordinates of upper left corner\n"
                                   "are determined by press on the picture\n"
                                   "Coordinates of bottom right corner\n"
                                   "are determined by mouse release");
        label->setStyleSheet("color: red;");
        layout->addWidget(label, 1, 0);
    }
}

void Dialog::RectangleAndColorDialog()
{
    QPushButton* but1 = new QPushButton("Old color");
    QPushButton* but2 = new QPushButton("New color");

    QLabel* lb1 = new QLabel("Choose rectangle color:");
    QLabel* lb2 = new QLabel("Choose new rectangle color");
    layout->addWidget(lb1, 0, 0);
    layout->addWidget(but1, 0, 1);
    layout->addWidget(lb2, 1, 0);
    layout->addWidget(but2, 1, 1);
    layout->addWidget(pcmdOk, 2, 0);
    layout->addWidget(pcmdCancel, 2, 1);
    resize(400, 100);

}

int Dialog::getSquareLineSize()
{
    return line_size->text().toInt();
}

int Dialog::getLineThickness()
{
    return thick->text().toInt();
}

int Dialog::getRgbNumber1()
{
    return red_box->currentIndex();
}

int Dialog::getRgbNumber2()
{
    return green_box->currentIndex();
}

int Dialog::getRgbNumber3()
{
    return blue_box->currentIndex();
}

int Dialog::getLeftXCoordinate()
{
    return x_left_coord->text().toInt();
}

int Dialog::getLeftYCoordinate()
```

```
{
    return y_left_coord->text().toInt();
}

int Dialog::getRightXCoordinate()
{
    return x_right_coord->text().toInt();
}

int Dialog::getRightYCoordinate()
{
    return y_right_coord->text().toInt();
}

int Dialog::getCircleRadius()
{
    return radius->text().toInt();
}

int Dialog::getDegree()
{
    return degree->currentIndex();
}

int Dialog::getRedValueOfBW()
{
    return red_bw->text().toInt();
}

int Dialog::getGreenValueOfBW()
{
    return green_bw->text().toInt();
}

int Dialog::getBlueValueOfBW()
{
    return blue_bw->text().toInt();
}
```

- **image.cpp**

```
#include "image.h"


Image::Image()
{
    PixmapIndex = 0;
    OldPixmaps = new PIXEL**[50];
}

Image::~Image()
{
    for (int i = 0; i < PixmapIndex; i++)
        delete [] OldPixmaps[i];
    delete [] OldPixmaps;

    for (int i = 0; i < max; i++)
        delete [] pixels[i];
    delete [] pixels;
}

int Image::load(const char* fname)
{
    FILE *f = fopen(fname, "r+b");
```

```
    fread(&bfh, sizeof(bfh), 1, f);
    fread(&bih, sizeof(bih), 1, f);

    if (bfh.bfReserved1 != 0 ||
            (bih.biSize != 40 && bih.biSize != 108 && bih.biSize != 124) ||
            bih.biWidth < 1 || bih.biWidth > 10000 ||
            bih.biHeight < 1 || bih.biHeight > 10000 ||
            bih.biCompression != 0 ||
            bih.biBitCount != 24)
        return -1;

    size_t padding = bih.biWidth % 4;
    max = bih.biHeight;
    if (max < bih.biWidth)
        max = bih.biWidth;

    pixels = new PIXEL*[max];
    for (int i = 0; i < max; i++)
        pixels[i] = new PIXEL [max + static_cast<long>(padding)];

    fseek(f, static_cast<long>(bfh.bfOffBits), SEEK_SET);
    for(int i = 0; i < bih.biHeight; i++) {
        for (int j = 0; j < bih.biWidth; j++)
            fread(&pixels[bih.biHeight - 1 - i][j], 3, 1, f);
        fread(&pixels[bih.biHeight - 1 - i][bih.biWidth], padding, 1, f);
    }

    fclose(f);
    return 1;
}

int Image::save(const char *fname)
{
    FILE* f;
    f = fopen(fname, "w+b");
    fwrite(&bfh, sizeof(BITMAPFILEHEADER), 1, f);
    fwrite(&bih, sizeof(BITMAPINFOHEADER), 1, f);

    size_t padding = bih.biWidth % 4;
    fseek(f, static_cast<long>(bfh.bfOffBits), SEEK_SET);
    for (int i = 0; i < bih.biHeight; i++)
    {
        for (int j = 0; j < bih.biWidth; j++)
            fwrite(&pixels[bih.biHeight - 1 - i][j], 1, 3, f);
        fwrite(&pixels[bih.biHeight - 1 - i][bih.biWidth], padding, 1, f);
    }

    fclose(f);
    return 1;
}

void Image::create()
{
    memset(&bih, 0, sizeof(bih));
    memset(&bfh, 0, sizeof(bfh));

    bih.biWidth = 1280;
    bih.biHeight = 720;
    bih.biSize = sizeof(bih);
    bih.biPlanes = 1;
    bih.biBitCount = 24;
    bfh.bfOffBits = 14 + sizeof(bih);
    bfh.bfType = 0x4d42;
    bfh.bfSize = static_cast<DWORD>(54 + bih.biHeight * bih.biWidth);
```

```
    for (int i = 0; i < bih.biHeight; i++) for (int j = 0; j < bih.biWidth; j++)
        pixels[i][j] = {255, 255, 255};
}


QPixmap Image::toQPixmap(int f)
{
    QPixmap pix;
    QImage* pict = new QImage(bih.biWidth, bih.biHeight, QImage::Format_RGB888);

    for (int i = 0; i < bih.biHeight; i++) for (int j = 0; j < bih.biWidth; j++)
    {
        QColor color(pixels[i][j].r, pixels[i][j].g, pixels[i][j].b);
        pict->setPixel(j, i, color.rgb());
    }
    pix.convertFromImage(*pict, nullptr);

    if (f)
        LoadOldPixmap();

    return pix;
}

void Image::LoadOldPixmap()
{
    if (PixmapIndex == 49)
    {
        ShiftOldPixmaps();
        PixmapIndex -= 1;
    }

    PixmapIndex += 1;
    OldPixmaps[PixmapIndex] = new PIXEL*[max];
    for (int i = 0; i < max; i++)
        OldPixmaps[PixmapIndex][i] = new PIXEL[max];

    for (int i = 0; i < bih.biHeight; i++) for (int j = 0; j < bih.biWidth; j++)
        OldPixmaps[PixmapIndex][i][j] = pixels[i][j];

}

void Image::ShiftOldPixmaps()
{
    for (int i = 0; i > 49; i++)
       OldPixmaps[i] = OldPixmaps[i + 1];
}

QPixmap Image::toOldQPixmap()
{
    if (PixmapIndex - 1 <= 0)
        return QPixmap("");
    PixmapIndex -= 1;

    for (int i = 0; i < bih.biHeight; i++) for (int j = 0; j < bih.biWidth; j++)
         pixels[i][j] = OldPixmaps[PixmapIndex][i][j];

    return toQPixmap(0);

}

QString Image::info() const
{
    QString information;
    QString n = QString::number(static_cast<double>(bfh.bfSize) / 1048576, 'g', 3);
    information.sprintf("File format: .bmp\nFile resolution: %ld x %ld\n", bih.biWidth,
bih.biHeight);
```

```
    information += "Size on disk: " + n + " Mb\n";
    information += "Bit count: " + QString::number(bih.biBitCount);
    return information;
}

void Image::SwapCoordinates()
{
    long tmp = bih.biHeight;
    bih.biHeight = bih.biWidth;
    bih.biWidth = tmp;
}

long Image::getWidth()
{
    return bih.biWidth;
}

long Image::getHeight()
{
    return bih.biHeight;
}

PIXEL** Image::getPixelsArr()
{
    LoadOldPixmap();
    return pixels;
}
```

- **main.cpp**

```
#include "mainwindow.h"
#include <QMessageBox>
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    app.setWindowIcon(QIcon("D:/qt/course work/coursework/Icons/paint.png"));
    MainWindow w;
    w.setWindowTitle("Course work");
    w.show();
    return app.exec();
}
```

- **mainwindow.cpp**

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    label = new DragableFrame;
    CreateCircleMenu();
    CreateRotateMenu();
    SetIcons();

}

MainWindow::~MainWindow()
{
    delete ui;
    delete good_filename;
}
```

```
void MainWindow::CreateCircleMenu()
{
    QMenu* menu = new QMenu;
    QAction* bt1 = new QAction("Radius and center coordinates");
    QAction* bt2 = new QAction("Square coordinates");
    bt1->setIcon(QIcon("D:/qt/course work/coursework/Icons/center_circle.png"));
    bt2->setIcon(QIcon("D:/qt/course work/coursework/Icons/square_and_circle.png"));
    menu->addAction(bt1);
    menu->addAction(bt2);
    connect(bt1, SIGNAL(triggered()), this,
SLOT(on_actionRadius_and_center_coordinates_triggered()));
    connect(bt2, SIGNAL(triggered()), this,
SLOT(on_actionSquare_coordinates_triggered()));
    ui->actionCircle_2->setMenu(menu);
}

void MainWindow::CreateRotateMenu()
{
    QMenu* menu = new QMenu;
    QAction* bt1 = new QAction("Full picture");
    QAction* bt2 = new QAction("Part of picture");
    bt1->setIcon(QIcon("D:/qt/course work/coursework/Icons/rotate_full.png"));
    bt2->setIcon(QIcon("D:/qt/course work/coursework/Icons/rotate_part.png"));
    menu->addAction(bt1);
    menu->addAction(bt2);
    connect(bt1, SIGNAL(triggered()), this, SLOT(on_RotateFullPicture_triggered()));
    connect(bt2, SIGNAL(triggered()), this, SLOT(on_RotatePartOfPicture_triggered()));
    ui->actionRotate->setMenu(menu);
}

void MainWindow::SetIcons()
{
    ui->actionRgb->setIcon(QIcon("D:/qt/course work/coursework/Icons/rgb.png"));
    ui->actionCircle_2->setIcon(QIcon("D:/qt/course work/coursework/Icons/circle.png"));
    ui->actionSquare->setIcon(QIcon("D:/qt/course work/coursework/Icons/square.png"));
    ui->actionStep_back_2->setIcon(QIcon("D:/qt/course
work/coursework/Icons/step_back.png"));
    ui->actionRotate->setIcon(QIcon("D:/qt/course work/coursework/Icons/rotate.png"));
    ui->actionFill_color->setIcon(QIcon("D:/qt/course
work/coursework/Icons/fill_color.jpg"));
    ui->actionLine_thickness->setIcon(QIcon("D:/qt/course
work/coursework/Icons/thickness.png"));
    ui->actionChoosing_color->setIcon(QIcon("D:/qt/course
work/coursework/Icons/line_color.jpg"));
    ui->actionOpen->setIcon(QIcon("D:/qt/course work/coursework/Icons/open_file.png"));
    ui->actionSave->setIcon(QIcon("D:/qt/course work/coursework/Icons/save_file.png"));
    ui->actionCreate->setIcon(QIcon("D:/qt/course
work/coursework/Icons/create_file.png"));
    ui->actionInfo_about_File->setIcon(QIcon("D:/qt/course
work/coursework/Icons/file_info.png"));
    ui->actionAbout_program_2->setIcon(QIcon("D:/qt/course
work/coursework/Icons/info.png"));
    ui->actionBW_component->setIcon(QIcon("D:/qt/course work/coursework/Icons/bw.png"));
}

void MainWindow::on_actionOpen_triggered()
{
    filename = QFileDialog::getOpenFileName(this, "Choose File",
tr("C:/Users/denis/Desktop"), tr("*.bmp"));
    if (!filename.isEmpty() && !LoadFile(filename))
    {
        QMessageBox::StandardButton reply = QMessageBox::question(this, "Info",
                                                                  "Do you want to try
again?");
        if (reply == QMessageBox::Yes) on_actionOpen_triggered();
```

```
    }
}

int MainWindow::LoadFile(QString filename)
{
    good_filename = new char[filename.length() + 1];
    std::strcpy(good_filename, filename.toLatin1().constData());

    int f = label->getImg()->load(good_filename);
    if (f <= 0)
    {
        QMessageBox::warning(this, "Error", "I can't open this file or his type is
uncorrect!");
        return 0;
    }

    label->resize(label->getImg()->getWidth(), label->getImg()->getHeight());
    label->getPnt()->LoadSize(label->getImg()->getWidth(), label->getImg()-
>getHeight());
    label->setPixmap(label->getImg()->toQPixmap(1));
    Scroll();

    return 1;
}

void MainWindow::Scroll()
{
    label->setBackgroundRole(QPalette::Base);
    label->setSizePolicy(QSizePolicy::Ignored, QSizePolicy::Ignored);
    label->setScaledContents(true);

    scrollArea = new QScrollArea;
    scrollArea->setWidget(label);
    setCentralWidget(scrollArea);
}

void MainWindow::on_actionCreate_triggered()
{
    label->getImg()->create();
    label->setPixmap(label->getImg()->toQPixmap(1));
}

void MainWindow::on_actionSave_triggered()
{
    QString fileName = QFileDialog::getSaveFileName(this, "Сохранить как...", "./",
"*.bmp");
    if (fileName.isNull()) return;
    std::strcpy(good_filename, fileName.toLatin1().constData());
    for (unsigned int i = 0; i < std::strlen(good_filename); i++) if (good_filename[i]
== '\\') good_filename[i] = '/';
    label->getImg()->save(good_filename);
}

void MainWindow::on_actionAbout_program_2_triggered()
{
    QMessageBox::information(this, "Info", "This program was created by Denis Ptuhov
using the QtCreator(v. 4.8.2)\n"
                                           "HotKeys:\n"
                                           "\t1) Ctrl + O - Open File\n"
                                           "\t2) Ctrl + S - Save File\n"
                                           "\t3) Ctrl + I - Information about file\n"
                                           "\t4) Ctrl + R - Create File");
}

void MainWindow::on_actionInfo_about_File_triggered()
{
```

```
    QMessageBox::information(this, "File info", label->getImg()->info());
}

void MainWindow::on_actionChoosing_color_triggered()
{
    chosen_color = QColorDialog::getColor(Qt::black, this, "Choosing color");
    if (chosen_color.isValid())
        label->getPnt()->UpdateColor(chosen_color);

}

void MainWindow::on_actionSquare_triggered()
{
    Dialog dialog(1);
    if (QDialog::Accepted == dialog.exec())
    {
        int n = dialog.getSquareLineSize();
        if (n <= 0)
        {
            QMessageBox::warning(this, "Wrong data", "Given coordinates is uncorrect");
            return;
        }

        label->getPnt()->LoadPixels(label->getImg()->getPixelsArr());
        label->activateDoingSquare(n);
    }
}

void MainWindow::on_actionRgb_triggered()
{
    Dialog dialog(2);
    if (QDialog::Accepted == dialog.exec())
    {
        int n = dialog.getRgbNumber1();
        checkRgb(n, 0);
        n = dialog.getRgbNumber2();
        checkRgb(n, 1);
        n = dialog.getRgbNumber3();
        checkRgb(n, 2);

        label->setPixmap(label->getImg()->toQPixmap(1));

    }
}

void MainWindow::checkRgb(int n, int n_color)
{
    label->getPnt()->LoadPixels(label->getImg()->getPixelsArr());

    if (n == 1)
        label->getPnt()->CreateRgb(n_color, 0);
    else if (n == 2)
        label->getPnt()->CreateRgb(n_color, 255);
}

void MainWindow::on_actionRadius_and_center_coordinates_triggered()
{
    Dialog dialog(3);
    if (QDialog::Accepted == dialog.exec())
    {
        int n = dialog.getCircleRadius();
        if (n <= 0)
        {
            QMessageBox::warning(this, "Wrong data", "Given coordinates is uncorrect");
            return;
        }
```

```
        label->getPnt()->LoadPixels(label->getImg()->getPixelsArr());
        label->activateDoingCircle(n);
    }
}

void MainWindow::on_actionSquare_coordinates_triggered()
{
    Dialog dialog(4);
    if (QDialog::Accepted == dialog.exec())
    {
        QPoint start_point(dialog.getLeftXCoordinate(), dialog.getLeftYCoordinate());
        QPoint current_point(dialog.getRightXCoordinate(),
dialog.getRightYCoordinate());

        if (current_point.y() - start_point.y() != current_point.x() - start_point.x()
||
            start_point.x() > label->getImg()->getWidth() || start_point.x() <= 0 ||
            start_point.y() > label->getImg()->getHeight() || start_point.y() <= 0 ||
            current_point.x() > label->getImg()->getWidth() || current_point.x() <= 0 ||
            current_point.y() > label->getImg()->getHeight() || current_point.y() <= 0)
        {
            QMessageBox::warning(this, "Wrong data", "Given coordinates is uncorrect");
            return;
        }
        label->getPnt()->LoadPixels(label->getImg()->getPixelsArr());
        label->getPnt()->CreateCircleUsingSquare(start_point, current_point);
        label->setPixmap(label->getImg()->toQPixmap(1));
    }
}

void MainWindow::on_actionLine_thickness_triggered()
{
    Dialog dialog(5);
    if (QDialog::Accepted == dialog.exec())
    {
        int n = dialog.getLineThickness();
        if (n <= 0)
        {
            QMessageBox::warning(this, "Wrong data", "Given value is uncorrect");
            return;
        }

        label->getPnt()->UpdateThickness(n);
    }
}

void MainWindow::on_actionFill_color_triggered()
{
    QColor fill_color = QColorDialog::getColor(Qt::white, this, "Choosing fill color");

    if (!fill_color.isValid())
        label->getPnt()->disconnectFill();
    else
        label->getPnt()->UpdateFillColor(fill_color);
}

void MainWindow::on_RotateFullPicture_triggered()
{
    Dialog dialog(6);
    if (dialog.exec() == QDialog::Accepted)
    {
        rt.LoadPixels(label->getImg()->getPixelsArr());
        rt.LoadSize(label->getImg()->getWidth(),
                    label->getImg()->getHeight());
```

```
            int n = dialog.getDegree();
            switch(n)
            {
                case 1:
                    rt.RotateOn90Full();
                    label->getImg()->SwapCoordinates();
                    label->resize(label->getImg()->getWidth(),
                                  label->getImg()->getHeight());
                    break;
                case 2:
                    rt.RotateOn180Full();
                    break;
                case 3:
                    rt.RotateOn270Full();
                    label->getImg()->SwapCoordinates();
                    label->resize(label->getImg()->getWidth(),
                                  label->getImg()->getHeight());
                    break;
            }
        }

        label->getPnt()->LoadPixels(label->getImg()->getPixelsArr());
        label->getPnt()->LoadSize(label->getImg()->getWidth(),
                                  label->getImg()->getHeight());
        label->setPixmap(label->getImg()->toQPixmap(1));
    }

    void MainWindow::on_RotatePartOfPicture_triggered()
    {
        Dialog dialog(7);
        if (dialog.exec() == QDialog::Accepted)
            label->activateDoingRotate(dialog.getDegree());

    }

    void MainWindow::on_actionStep_back_2_triggered()
    {
        QPixmap tmp = label->getImg()->toOldQPixmap();
        if (!tmp.isNull())
             label->setPixmap(tmp);
    }

    void MainWindow::on_actionBW_component_triggered()
    {
        label->getPnt()->LoadPixels(label->getImg()->getPixelsArr());
        label->getPnt()->LoadSize(label->getImg()->getWidth(),
                                  label->getImg()->getHeight());
        label->getPnt()->CreateBW();
        label->setPixmap(label->getImg()->toQPixmap(0));
    }

    void MainWindow::on_actionSmallest_rectangle_triggered()
    {
        Dialog dialog(8);
        if (dialog.exec() == QDialog::Accepted)
        {

        }
    }
```

- **painter.cpp**

```
#include "painter.h"
```

```
Painter::Painter()
{

}

void Painter::LoadPixels(PIXEL **new_pixels)
{
    pixels = new_pixels;
}

void Painter::LoadSize(int new_width, int new_height)
{
    width = new_width;
    height = new_height;
}

void Painter::CreateSquare(QPoint start_point, QPoint current_point)
{
    int x1 = start_point.x(),
        y1 = start_point.y(),
        x2 = current_point.x(),
        y2 = current_point.y();
    double k = static_cast<double>(y2 - y1) / static_cast<double>(x2 - x1);
    double b = static_cast<double>(y1) - k * x1;
    double k2 = -k;
    double b2 =  static_cast<double>(y1) - k2 * x2;

    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
    {
        if (fill && i > y1 && i < y2
                && j > x1 && j < x2)
            drawPixel(i, j, fill_color);

        if ((abs(i - j * k - b) <= thickness * 2 || abs(i - j * k2 - b2) <= thickness * 2)
                && i >= y1 && i <= y2
                && j >= x1 && j <= x2)
            drawPixel(i, j, CurrentLineColor);

        if ((abs(i - y1) <= thickness || abs(i - y2) <= thickness) && j >= x1 && j <= x2)
            drawPixel(i, j, CurrentLineColor);

        if ((abs(j - x1) <= thickness || abs(j - x2) <= thickness) &&
            i >= y1 - thickness && i <= y2 + thickness)
            drawPixel(i, j, CurrentLineColor);
    }
}

void Painter::CreateCircleUsingRadius(QPoint center, int radius)
{
    int x = center.x(),
        y = center.y();
    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
    {
        if (fill && (j - x)*(j - x) + (i - y)*(i - y) - radius * radius < 0)
            drawPixel(i, j, fill_color);

        if (abs((j - x)*(j - x) + (i - y)*(i - y) - radius * radius) <= radius * 3 *
thickness)
            drawPixel(i, j, CurrentLineColor);
    }
}

void Painter::CreateCircleUsingSquare(QPoint left, QPoint right)
{
    int a = right.x() - left.x();
```

```
    int radius = a / 2;
    QPoint center(left.x() + radius, left.y() + radius);
    CreateCircleUsingRadius(center, radius);
}

void Painter::UpdateColor(QColor new_color)
{
    CurrentLineColor = new_color;
}

void Painter::UpdateFillColor(QColor new_color)
{
    fill = true;
    fill_color = new_color;
}

void Painter::UpdateThickness(int new_thickness)
{
    thickness = new_thickness;
}

void Painter::disconnectFill()
{
    fill = false;
}

void Painter::drawPixel(int x, int y, QColor cur_color)
{
    pixels[x][y].r = static_cast<unsigned char>(cur_color.red());
    pixels[x][y].g = static_cast<unsigned char>(cur_color.green());
    pixels[x][y].b = static_cast<unsigned char>(cur_color.blue());
}

void Painter::CreateRgb(int n_color, int n)
{
    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
    {
        if (!n_color)
            pixels[i][j].r = static_cast<unsigned char>(n);
        else if (n_color == 1)
            pixels[i][j].g = static_cast<unsigned char>(n);
        else
            pixels[i][j].b = static_cast<unsigned char>(n);
    }

}

void Painter::CreateBW()
{
    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
    {
        unsigned char n = static_cast<unsigned char>((pixels[i][j].r + pixels[i][j].b +
pixels[i][j].g) / 3);
        pixels[i][j] = {n, n, n};
    }
}
```

- **rotate.cpp**

```
#include "painter.h"


Painter::Painter()
{
```

```
}

void Painter::LoadPixels(PIXEL **new_pixels)
{
    pixels = new_pixels;
}

void Painter::LoadSize(int new_width, int new_height)
{
    width = new_width;
    height = new_height;
}

void Painter::CreateSquare(QPoint start_point, QPoint current_point)
{
    int x1 = start_point.x(),
        y1 = start_point.y(),
        x2 = current_point.x(),
        y2 = current_point.y();
    double k = static_cast<double>(y2 - y1) / static_cast<double>(x2 - x1);
    double b = static_cast<double>(y1) - k * x1;
    double k2 = -k;
    double b2 =  static_cast<double>(y1) - k2 * x2;

    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
    {
        if (fill && i > y1 && i < y2
                && j > x1 && j < x2)
            drawPixel(i, j, fill_color);

        if ((abs(i - j * k - b) <= thickness * 2 || abs(i - j * k2 - b2) <= thickness * 2)
                && i >= y1 && i <= y2
                && j >= x1 && j <= x2)
            drawPixel(i, j, CurrentLineColor);

        if ((abs(i - y1) <= thickness || abs(i - y2) <= thickness) && j >= x1 && j <= x2)
            drawPixel(i, j, CurrentLineColor);

        if ((abs(j - x1) <= thickness || abs(j - x2) <= thickness) &&
            i >= y1 - thickness && i <= y2 + thickness)
            drawPixel(i, j, CurrentLineColor);
    }
}

void Painter::CreateCircleUsingRadius(QPoint center, int radius)
{
    int x = center.x(),
        y = center.y();
    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
    {
        if (fill && (j - x)*(j - x) + (i - y)*(i - y) - radius * radius < 0)
            drawPixel(i, j, fill_color);

        if (abs((j - x)*(j - x) + (i - y)*(i - y) - radius * radius) <= radius * 3 *
thickness)
            drawPixel(i, j, CurrentLineColor);
    }
}

void Painter::CreateCircleUsingSquare(QPoint left, QPoint right)
{
    int a = right.x() - left.x();
    int radius = a / 2;
    QPoint center(left.x() + radius, left.y() + radius);
    CreateCircleUsingRadius(center, radius);
```

```
}

void Painter::UpdateColor(QColor new_color)
{
    CurrentLineColor = new_color;
}

void Painter::UpdateFillColor(QColor new_color)
{
    fill = true;
    fill_color = new_color;
}

void Painter::UpdateThickness(int new_thickness)
{
    thickness = new_thickness;
}

void Painter::disconnectFill()
{
    fill = false;
}

void Painter::drawPixel(int x, int y, QColor cur_color)
{
    pixels[x][y].r = static_cast<unsigned char>(cur_color.red());
    pixels[x][y].g = static_cast<unsigned char>(cur_color.green());
    pixels[x][y].b = static_cast<unsigned char>(cur_color.blue());
}

void Painter::CreateRgb(int n_color, int n)
{
    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
    {
        if (!n_color)
            pixels[i][j].r = static_cast<unsigned char>(n);
        else if (n_color == 1)
            pixels[i][j].g = static_cast<unsigned char>(n);
        else
            pixels[i][j].b = static_cast<unsigned char>(n);
    }

}

void Painter::CreateBW()
{
    for (int i = 0; i < height; i++) for (int j = 0; j < width; j++)
    {
        unsigned char n = static_cast<unsigned char>((pixels[i][j].r + pixels[i][j].b +
pixels[i][j].g) / 3);
        pixels[i][j] = {n, n, n};
    }
}
```

- **mainwindow.ui**

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1019</width>
```

```
  <height>638</height>
 </rect>
</property>
<property name="windowTitle">
 <string>MainWindow</string>
</property>
<widget class="QWidget" name="centralWidget"/>
<widget class="QStatusBar" name="statusBar"/>
<widget class="QMenuBar" name="menuBar">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>1019</width>
   <height>26</height>
  </rect>
 </property>
 <widget class="QMenu" name="Tools">
  <property name="title">
   <string>Tools</string>
  </property>
  <addaction name="actionRgb"/>
  <addaction name="separator"/>
  <addaction name="actionBW_component"/>
  <addaction name="separator"/>
  <addaction name="actionSquare"/>
  <addaction name="separator"/>
  <addaction name="actionCircle_2"/>
  <addaction name="separator"/>
  <addaction name="actionRotate"/>
  <addaction name="separator"/>
  <addaction name="actionStep_back_2"/>
  <addaction name="separator"/>
  <addaction name="actionSmallest_rectangle"/>
 </widget>
 <widget class="QMenu" name="menu">
  <property name="title">
   <string>File</string>
  </property>
  <addaction name="separator"/>
  <addaction name="actionOpen"/>
  <addaction name="separator"/>
  <addaction name="actionCreate"/>
  <addaction name="separator"/>
  <addaction name="actionSave"/>
  <addaction name="separator"/>
 </widget>
 <widget class="QMenu" name="menuInfo">
  <property name="title">
   <string>Info</string>
  </property>
  <addaction name="actionInfo_about_File"/>
  <addaction name="separator"/>
  <addaction name="actionAbout_program_2"/>
 </widget>
 <widget class="QMenu" name="menuChoosing_color">
  <property name="title">
   <string>Tools format</string>
  </property>
  <addaction name="actionChoosing_color"/>
  <addaction name="separator"/>
  <addaction name="actionLine_thickness"/>
  <addaction name="separator"/>
  <addaction name="actionFill_color"/>
 </widget>
 <addaction name="menu"/>
```

```
 <addaction name="Tools"/>
 <addaction name="menuChoosing_color"/>
 <addaction name="menuInfo"/>
</widget>
<action name="actionOpen">
 <property name="text">
  <string>Open</string>
 </property>
 <property name="shortcut">
  <string>Ctrl+O</string>
 </property>
</action>
<action name="actionCreate">
 <property name="text">
  <string>Create</string>
 </property>
 <property name="shortcut">
  <string>Ctrl+R</string>
 </property>
</action>
<action name="actionSave">
 <property name="text">
  <string>Save</string>
 </property>
 <property name="shortcut">
  <string>Ctrl+S</string>
 </property>
</action>
<action name="actionAbout_program">
 <property name="text">
  <string>About program</string>
 </property>
</action>
<action name="actionRgb">
 <property name="text">
  <string>Rgb</string>
 </property>
</action>
<action name="actionReverse">
 <property name="text">
  <string>Reverse</string>
 </property>
</action>
<action name="actionCircle">
 <property name="text">
  <string>Circle</string>
 </property>
</action>
<action name="actionSquare">
 <property name="text">
  <string>Square</string>
 </property>
</action>
<action name="RotateFullPicture">
 <property name="text">
  <string>Full picture</string>
 </property>
</action>
<action name="RotatePartOfPicture">
 <property name="text">
  <string>Part of picture</string>
 </property>
</action>
<action name="actionInfo_about_File">
 <property name="text">
  <string>About file</string>
```

```
 </property>
</action>
<action name="actionAbout_program_2">
 <property name="text">
  <string>About program</string>
 </property>
</action>
<action name="actionChoosing_color">
 <property name="text">
  <string>Line color</string>
 </property>
</action>
<action name="actionLine_thickness">
 <property name="text">
  <string>Line thickness</string>
 </property>
</action>
<action name="actionFill_color">
 <property name="text">
  <string>Fill color</string>
 </property>
</action>
<action name="actionRadius_and_center_coordinates">
 <property name="text">
  <string>Radius and center coordinates</string>
 </property>
</action>
<action name="actionSquare_coordinates">
 <property name="text">
  <string>Square coordinates</string>
 </property>
</action>
<action name="actionStep_back">
 <property name="text">
  <string>Step back</string>
 </property>
 <property name="shortcut">
  <string>Ctrl+Z</string>
 </property>
</action>
<action name="actionCircle_2">
 <property name="text">
  <string>Circle</string>
 </property>
</action>
<action name="actionStep_back_2">
 <property name="text">
  <string>Step back</string>
 </property>
 <property name="shortcut">
  <string>Ctrl+Z</string>
 </property>
</action>
<action name="actionRotate">
 <property name="text">
  <string>Rotate</string>
 </property>
</action>
<action name="actionBW_component">
 <property name="text">
  <string>BW component</string>
 </property>
</action>
<action name="actionSmallest_rectangle">
 <property name="text">
  <string>Smallest rectangle</string>
```

```
   </property>
  </action>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
</ui>
```