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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP-файла

Студент гр. 8303 _____ Гришин К. И.

Преподаватель _____ Чайка К. В.

Санкт-Петербург
2019

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Гришин Константин

Группа 8303

Тема работы: Обработка BMP-файла

Исходные данные:
Требуется написать программу для обработки изображений формата BMP. Программа должна иметь класс для хранения изображения в памяти, а также содержать следующие методы обработки:
Рисование шестиугольника, копирование области, смена пикселей одного цвета на другой, рисование рамки вокруг изображения.

Содержание пояснительной записки:
Перечисляются требуемые разделы пояснительной записки (обязательны разделы «Содержание», «Введение», «Заключение», «Список использованных источников»)

Предполагаемый объем пояснительной записки:
Не менее 58 страниц.

Дата выдачи задания: 27.02.2019

Студент _____ Гришин К. И.

Преподаватель _____ Чайка К. В.

АННОТАЦИЯ

Целью данной работы является создание программы для обработки растровых изображений формата BMP. Для этого были созданы функции для загрузки изображения в оперативную память, сохранения и непосредственного изменения цвета пикселей, такие как рисование правильного шестиугольника, копирования области изображения, изменения пикселей одного цвета на другой и рисование рамки. Для удобного использования программы, к ней был разработан GUI позволяющий наглядно изменять параметры функционала. В работе проведено тестирование программы, а также приведен ее исходный код.

SUMMARY

The aim of this for is creating a program for processing bitmap images. For this purpose, function were created to load images into RAM, save and directly change tho color of pixels, such as drawing a regular hexagon, copying of image area, changing of pixels of one color to another, drawing a frame around the image. For convenient using of program, GUI was developed to it, which allows to visually change the parameters of functions. The program was tested. The source code is submitted in the annex.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Программа написана на языке C++ с использованием графического фреймфорка Qt, а также графической формой Qt Designer. Для работы с изображением создан класс Image. Графический интерфейс сделан на базе шаблонных элементов Qt.

ЦЕЛЬ И УСЛОВИЕ РАБОТЫ

Вариант 7.

Общие сведения:

- 24 бита на цвет.
- без сжатия.
- файл всегда соответствует формату BMP.
- 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

Программа должны реализовывать весь следующий функционал по обработке bmp-файла:

- Рисование правильного шестиугольника. Шестиугольник определяется:
  - координатами противоположных углов квадрата, в который вписан, или центром и радиусом окружности, в которую вписан
  - толщиной линии
  - цветом линии
  - шестиугольник может быть залит
- Копирование заданной области. Функционал определяется:
  - координатами противоположных углов прямоугольной области источника
  - координатами левого верхнего угла области назначения
- Заменяет все пиксели одного цвета на пиксели другого цвета. Функционал определяется:
  - Цвет, который требуется заменить
  - Цвет на который требуется заменить
- Сделать рамку в виде узора. Рамка определяется:
  - Узором
  - Цветом
  - Шириной

1. КЛАСС MAINWINDOW И ЕГО СЛОТЫ

1.1. Определение класса MainWindow

Класс содержит поля Image *img и Grapher *grapher, которые являются указателями на пользовательские классы, первый для хранения изображения, второй для его отображения; bool fill и int thick, которые отвечают за заливку и толщину соответственно; Qcolor fore и Qcolor back, которые отвечают за основной и дополнительный цвет соответственно; Ui::Mainwindow *ui, поле которые отвечает за непосредственно интерфейс основного окна, созданный в Qt Designer.

Слоты класса, вызываемые сигналами Grapher для работы с полем Image:

on_new_file_triggered, *on_save_file_triggered*, *on_open_file_triggered* — создают новый, сохраняют и открывают файл соответственно. Вызываются по нажатию опции в меню *«Файл»*.

on_about_image_triggered, *on_about_triggered* — показывают информацию о текущем изображении и программе соответственно. Вызываются из опции в меню *«Справка»*.

on_Fill_stateChanged, *on_Width_valueChanged* — отвечают за смену значения полей *fill* и *thick* соотвественно. Данные опции расположены в нижней функциональной части, после всех функции меню.

on_foreground_clicked, *on_background_clicked* — отвечают за смену значений полей *fore* и *back* соответственно. Вызываются по нажатию на цветные квадраты, расположенные между основным функционалом и опциями толщины с заливкой.

show_coordinates — отвечает за отображение координат расположения курсора на холсте. Вызывается каждый раз при перемещении мыши по холсту.

on_Hexagon_clicked — отвечает за включение функции рисования шестиугольника.

on_Copy_clicked — отвечает за включение функции по копированию заданной области.

on_Recolor_clicked — отвечает за включение функции по смене цвета.

on_Frame_clicked — отвечает за рисование рамки, вызывает меню с выбором узора.

on_Pen_clicked — отвечает за включение пера.

Все выше описанные слоты вызываются по нажатию на функциональные кнопки *«Шестиугольник»*, *«Копирование»*, *«Смена цвета»*, *«Рамка»*, *«Перо»* соответственно.

Слоты вызываемые сигналами из *grapher* и изменяющие на объект класса *Image*:

pen — риисует мелкие отрезки между сигналами mouse_move_event, посылаемыми из *grapher*.

hexagon — рисует шестиугольник, вызывается сигналом из mouse_release_event, используя параметры, полученные при перемещении курсора.

copy — копирует ранее выделенную выделенную область, вызывается сигналом из *mouse_press_event*, использует параметры полученные при предыдущем выделении области.

recolor — меняет все цвета основного цвета на дополнительный в выделенной области, вызывается сигналом из *mouse_release_event*, используя параметры полученные при перемещении курсора.

Класс *MainWindow* также содержит переопределенные конструктор и дестркутор.

1.2. Конструктор и деструктор класса MainWindow

Конструктор восстанавливает объекты из формы *ui*.

Создаются объекты *Grapher* и *Image*, первый является наследником объектом Qt и располагается на главном окне.

Настраиваются цвета палитры.

Соединяются сигналы, посылаемые ивентами мыши из *grapher* со слотами из *MainWindow*.

В деструкторе освобождается память занимаемая объектами *Grapher*, *Image* и *Ui*.

1.3. Сохранение изображения, загрузка и создание нового

on_save_file_triggered — функция запускает стандартное диалоговое окно *QFileDialog::getSaveFileName*, которое позволяет выбрать нужную директорию для файла, а также его название. Далее полный путь файла используется в методе *«saveImage»* класса *Image*.

on_load_file_triggered — функция запускает стандартное диалоговое окно *QFileDialog::getOpenFileName*, аналогичное тому, что описано в предыдущем абзаце. Полученный путь до файла затем используется в методе *«loadImage»* класса *Image*. Метод возвращает код ошибки, по которому обрабатываются исключительные ситуации и по надобности вызываются окна сообщающие об ошибках загрузки.

on_new_file_triggered — Функция создает объект созданного класса *NewImageDialog*, который наследуется от QDialog. В созданном диалоговом окне определяются значения ширины и высоты пустого холста. Эти значения используются в методе *«newImage»* класса *Image*.

1.4. Слоты, вызываемые нажатием кнопок

on_Hexagon_clicked, *on_Copy_clicked*, *on_Recolor_clicked*, *on_Pen_clicked* — проверяют объект класса *Image* на то, загружено ли в программу изображение. Если не загружено, то программа сообщает о том, что требуется загрузка изображения. В ином случае в объект класса *Grapher* устанавливается функция *HEXAGON*, *COPY*, *RECOLOR* и *PEN* соответственно, при этом обнуляется флаг копирования *«copy»* в *grapher*.

on_Frame_clicked — создает новый объект созданного класса *CreateFrameDialog*, который дает возможность пользователю выбрать узор и сделать небольшие коррективы в его форме. Затем с помощью оператора *switch*, вызывается метод *«createFrame»* класса *Image* с определенным типом рисунка.

on_foreground_clicked, *on_background_clicked* — вызывают стандартное диалоговое окно *QColorDialog::getColor*, которое позволяет получить цвет из палитры. Этот цвет устанавливается в поля *fore* и *back* объекта класса *MainWindow* соответственно. В этот же цвет окрашиваются кнопки, путем смены их палитры.

on_Fill_stateChanged, *on_Width_valueChanged* — Устанавливают значение полей *fill* и *thick* объекта класса *MainWindow*.

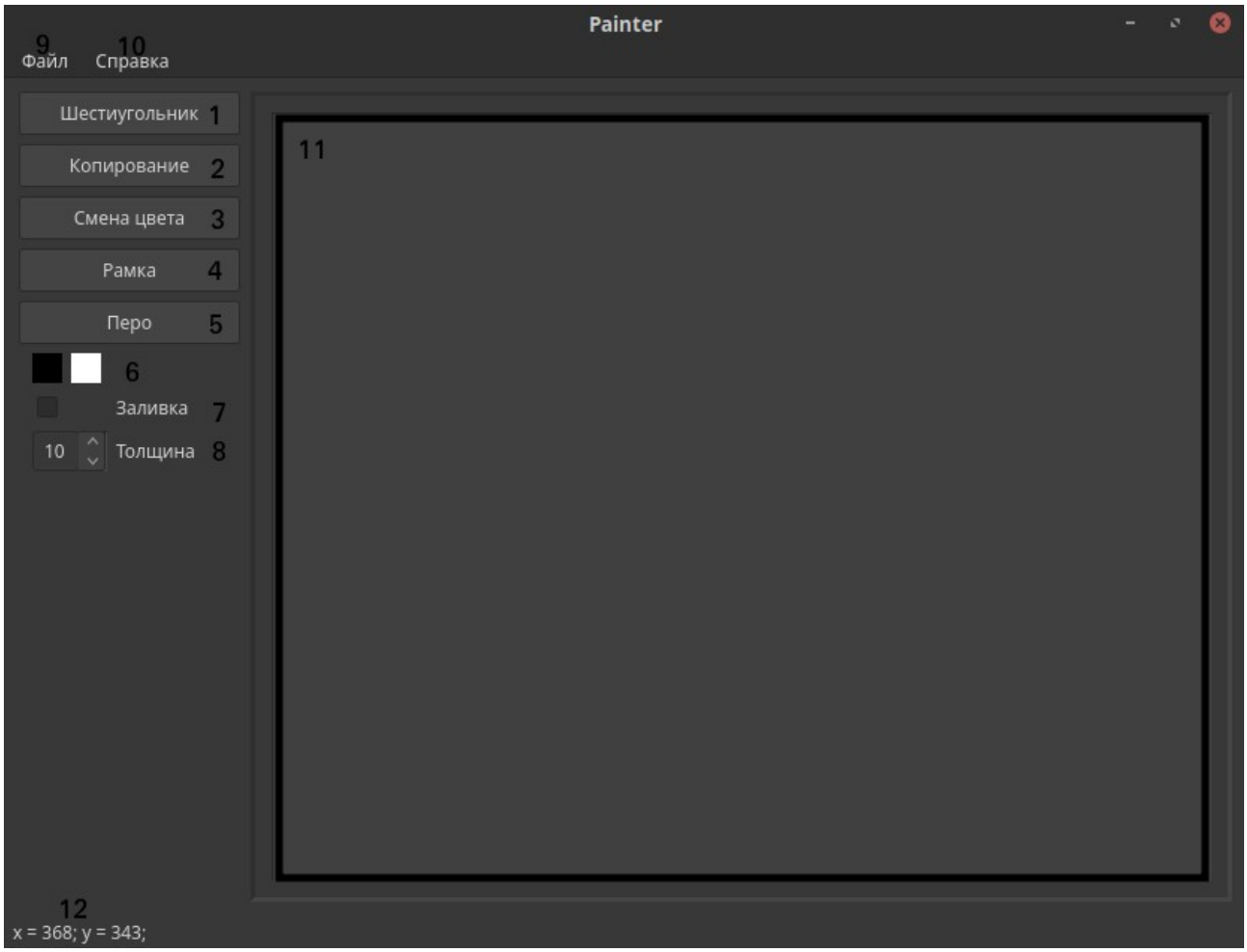
on_about_triggered, *on_about_image_triggered* — Информативные блоки, первый показывает информацию о программе (a.k.a. инструкцию), второй показывает информацию о текущем изображении, а именно ширину, высоту и размер.

1.5. Слоты вызываемые сигналами из grapher

pen, *hexagon*, *copy*, *recolor* — вызывают методы «setPart», *«paintHexagon»*, *«copyRect»*, *«recolorRect»,* которые изменяют матрицу пикселей а затем обновляют *pixmap* в *grapher*.

show_coordinates — вызывается при любых манипуляциях мышью, определяет координаты курсора относительно верхнего левого угла холста и выводит их в статусбар.

1.6. Внешний вид главного окна.



1 — *on_Hexagon_clicked*
2 — *on_Copy_clicked*
3 — *on_Recolor_clicked*
4 — *on_Frame_clicked*
5 — *on_Pen_clicked*
6 — *on_foreground_clicked (слева) и on_background_clicked (справа)*
7 — *on_Fill_stateChanged*
8 — *on_Width_valueChanged*
9 — *on_save_file_triggered, on_new_file_triggered, on_open_file_triggered*
10 — *on_about_image_triggered, on_about_triggered*
11 — *grapher*
12 — show_coordinates

2. КЛАСС GRAPHER И ЕГО СИГНАЛЫ

2.1. Определение класса Grapher

Поля класса:

*QGraphicsScene *scene* — сцена на которую накладываются изображение и элементы рисования (области действия функций).

coordinate start — объект хранящий координату, в которой была нажата мышь.

coordinate finish — объект хранящий координату в которую записывается положение мыши, когда она отпущена.

int func — переменная хранящая значение текущей функции, *int copy* — флаг для смены функционала при копировании (0 — область выделяется, 1 — область вставляется).

*QGraphicsItemGroup *object*, *QGraphicsItemGroup *hexagon* — группы элементов (прямых), которые динамически меняются при перемещении мыши.

double radius — переменная, в которой хранится радиус шестиугольника, *double angle* — переменная, в которой хранится угол поворота шестиугольника.

QPixmap — тип данных, который позволяет хранить отображаемую матрицу пикселей.

bool drawer — флаг отвечающий за то, нажата ли мышь.

Методы класса:

void update(QPixmap) - обновляет Pixmap, которая расположена на сцене.

void freeGroup(QGraphicsItemGroup)* - удаляет все элементы из группы.

Конструктор и деструктор.

Сигналы класса:

mouse_track_signal — сигнал, который сообщает в *MainWindow* координаты, запускается при любых манипуляциях с мышью.

pen_signal — сигнал, который запускается в *mouse_move_event*, позволяя тем самым рисовать на холсте мелкие отрезки, между посылаемыми сигналами.

square_signal — сигнал, который сообщает в *MainWindow* координаты точки нажатия мыши и ее отпуска.

copy_signal — сигнал, выпускаемый из *mouse_press_event*, который сообщает в *MainWindow* координаты предыдущего нажатия мыши и ее отпуска, а также текущего нажатия.

hexagon_signal — сигнал, сообщающий в *MainWindow* координаты нажатия мыши и расстояния вектора от нажатия до отпуска, а также угла, между вектором и горизонтальной прямой.

Также класс содержит такие слоты как *mouse_press_event*, *mouse_move_event*, *mouse_release_event*. Эти слоты отвечают за информацию, получаемые при манипуляциях с мышью.

2.2. Конструктор и деструктор класса

Конструктор создает объект класса *QGraphicsScene*, на котором располагаются изображения и прямые определяющие функционал программы. Определяется начало отсчета координатной плоскости путем установления выравнивания. В *Grapher* (a.k.a. *QGraphicsView*) устанавливается сцена. Создаются объекты objecet и hexagon, объявленные в полях класса. Данные объекты добавляются на сцену. Включается отслеживание мыши.

Деструктор же является пустым.

2.3. Методы и слоты класса

Методы:

update — удаляет *Pixmap* расположенную на сцене и ставит новую.

freeGroup — проходится по всем элементам сцены и удаляет те, которые принадлежат группе

Слоты:

mouse_press_event — обнуляются поля *radius* и *angle*, создается объект класса QPoint, в который записываются координаты текущего положения мыши и посылается сигнал *mouse_track_signal*. Затем проверяется была ли уже выделена область функцией COPY, и если выделена, то посылается соответствующий сигнал копирования. Включается флаг нажатой мыши и полям *start* и *finish* присваивается значение текущего положения мыши.

mouse_move_event — создается объект класса *QPoint*, в который записывается текущее положение мыши и посылается сигнал *mouse_track_signal*. Весь остальной функционал выполняется только при условии, что мышь нажата.
Для пера сначала *finish* присваивается значение текущего положения мыши, затем посылается сигнал *pen_signal*, после чего в *start* записывается *finish*.
Для COPY и RECOLOR объекту *finish* присваивается значение текущего положения мыши, затем освобождается и пересоздается группа *object* (которая представляет из себя прямоугольник). Системному перу присваивается черный цвет и рисуется прямоугольник исходя из того, что текущие *start* и *finish* являются противоположными углами.
Для HEXAGON объекту *finish* присваивается значение текущего положения мыши, затем освобождается и пересоздается группа *hexagon* (которая представляет из себя правильный шестиугольник). Системному перу присваивается черный цвет, затем вычисляются значения радиус-вектора (*start*, *finish*) и угла между этим вектором и горизонталью. Затем циклом создается и соединяется шесть точек, которые просчитываются с помощью функций синуса и косинуса и смещаются на фазу, которой является ранее просчитанный угол.

mouse_release_event — если текущей функцией является COPY, то флагу *copy* присваивается значение 1, что говорит о том, что требуемая область выделена.

Затем создается объект класса *QPoint*, в который записывается положение текущей координаты. Полю *finish* присваивается значение только что полученной координаты. Затем освобождаются группы *hexagon* и *object*, флагу *drawer* присваивается значение 0, это означает, что мышь отпущена. Посылаются сигналы *pen_signal*, *square_signal*, *hexagon_signal*, в зависимости от текущей соответствующей функции.

3. КЛАСС IMAGE И ЕГО МЕТОДЫ

3.1. Определение класса Image

Поля класса:

BITMAPFILEHEADER bfh — структура, хранящая первый заголовок bmp файла

BITMAPINFOHEADER bih — структура, хранящая второй заголовок bmp файла

*RGB **rgb* — матрица, хранящая изображение путем использования структуры *rgb*, где есть три поля для хранения каждого канала.

Методы класса:

Функция содержит один геттер — *getMap*, который преобразовывает матрицу пикселей в объект *QPixmap* и возвращает его.

Манипуляции с файлом:

newImage — получает на вход ширину и высоту нового изображения, по которым просчитываются значения заголовков, выделяется память для поля *rgb** и заполняется пикселями белого цвета.

saveImage — получает на вход название файла, в который сохраняется изображения, в которое записывается сначала значения заголовков, затем постепенно каждый пиксель строки, начиная с нижней.

loadImage — получает на вход название файла, который требуется открыть. Вначале проверяется на возможность открытия этого файла, затем проверяется первый заголовок, после чего второй заголовок и потом постепенно загружаются пиксели в rgb пропуская при этом выравнивание.

Методы, действующие на матрицу rgb (непосредственно рисование)

floodFill — заливает область, начиная с заданного пикселя, ограниченную цветом, которое подается в функцию.

setPart — вставляет отрезок по двум координатам, на концах которого находятся квадраты.

drawLine — вставляет отрезок по двум координатам толщиной в один пиксель.

drawThickLine — рисует пучок отрезков (количество определяется толщиной), а на концах располагает окружности.

defCircle — рисует залитую окружность из заданной точки, заданного радиуса.

drawCircle — рисует окружность, с границей заданной толщины, присутствует возможность рисования залитой окружности.

defHexagon — рисует залитый правильный шестиугольник заданного цвета (задаются также центр, радиус, фаза)

drawHexagon — рисует шестиугольник с границей заданной толщины и заданного цвета, а также присутствует возможность заливки внутренней части шестиугольника. Также задаются центр, радиус и фаза.

copyRect — создается буффер заданного размера, в который копируется область относительно заданный координат. Затем содержимое буффера накладывается на изображение относительно точки, которая является верхним левым углом.

recolorRect — в прямоугольнике с заданными координатами один цвет заменяется другим.

createFrame — получает на вход цвет, тип узора, параметр узора. Создается буффер размером в исходное изображение и заливается заданным цветом. Каждый из пикселей в буффере имеет параметр наложения, от которого зависит какой именно пиксель будет перемещен в матрицу rgb. Для узора «обод» просто рисуется набор прямых относительно от границ изображения. Для оставшихся узоров создается еще один буффер, который просчитывает внешний вид одного элемента узора, затем замощается общий буффер, с которого полученная рамка переносится на основной холст.

3.2. Методы класса Image

getMap() — создается объект класса *QImage*, в который копируются все пиксели матрицы rgb. Затем возвращается матрица пикселей *QPixmap*, полученная из объекта *QImage*.

floodFill(int x, int y, QColor color) — создается объект структуры point, в который записываются координаты x, y. Затем создается стек и стандартной библиотеки C++. С использованием этого стека запускается алгоритм заливки пока не будет встречен цвет *color* или граница матрицы *rgb*.

setPart(int s_x, int s_y, int f_x, int f_y, QColor color, int thick) — создается два квадрата со стороной *thick* и центрами в точках s и f. Далее из каждой точки диагонали одного квадрата создается прямая в соответствующую точку другого квадрата.

drawLine(int s_x, int s_y, int f_x, int f_y, QColor color) — используется алгоритм Брезенхема с проверкой на выход за границу матрицы пикселей.

drawThickLine(int s_x, int s_y, int f_x, int f_y, int thick, QColor color) — выбирается вертикальная или горизонтальная прямая с центрами в точках s и f, в зависимости от того, вдоль какой координаты смещение больше, если x — вертикальная, если y — то горизонтальная. Затем из каждой точки прямой в s проводится прямая в соответствующую точку на прямой в f.

defCircle(int s_x, int s_y, double radius, QColor color) — в квадрате с центром в s и стороной **2radius** каждый пиксель проверяется и закрашивается цветом *color*, если он удовлетворяет условию $\mathbf{x^2 + y^2 <= radius^2}$.

drawCircle(int s_x, int s_y, double radius, int thick, QColor) — накладывает две окружности путем применения метода *defCircle* с радиусами *radius* и ***radius-thick***.

defHexagon(int s_x, int f_x, double radius, double angle, QColor color) — определяется шесть точек с помощью функций sin и cos, в каждой из которых есть сдвиг по фазе (*a.k.a angle*). Затем этот шестиугольник заливается.

drawHexagon(int s_x, int s_y, double radius, double angle, int thick, bool fill, QColor fore, QColor back) — Два шестиугольника разного цвета накладываются друг на друга, где первый радиуса *radius* и цвета *fore*, второй радиуса *radius* — *thick* и цвета *back*, при этом сдвиг по фазе одинаковый, а отображение меньшего шестиугольника зависит от переменной *fill*.

copyRect(int s_x, int s_y, int f_x, int f_y, int n_x, int n_y) — создается буффер размера разностей соответствующих координат, в который циклично записываются все пиксели из прямоугольника с противоположными углами s и f. Затем все пиксели вставляются относительно точки n, если они не выходят за границы матрицы *rgb*.

recolorRect(int s_x, int s_y, int f_x, int f_y, QColor old_color, QColor new_color) — в прямоугольнике с противоположными углами в s и f циклично каждый пиксель цвета *old_color* заменяется цветом *new_color*.

createFrame(QColor primary, int type, double parameter) — создается основной буффер размером в изображение и цветом *additional*, притом так, что каждый пиксель в нем является неотображаемым. Затем в зависимости от type начинается изменение определенных пикселей в основном буффере таким образом, что они становятся отображаемыми. Если узором являются *«ветви»* или *«крест»*, то создается дополнительный буффер, в котором отрисовывается один элемент рамки, которым потом замощается основной буффер. После того, как будет получен основной буффер с требуемыми пикселями, помеченными как *«отображаемые»*, этот буффер накладывается на основное изображение.

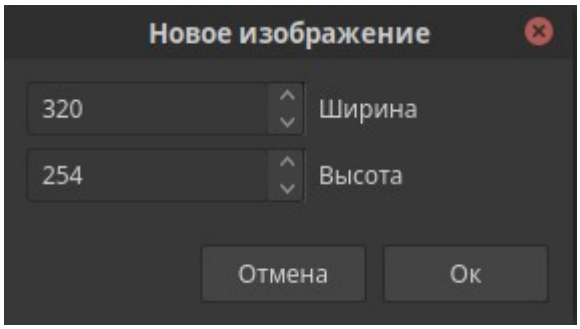
~Image() - дестурктор класса, который освобождает память, занимаемую ***rgb*.

4. ДОПОЛНИТЕЛЬНЫЕ ОКНА ВЗАИМОДЕЙСТВИЯ

4.1. NewImageDialog

Простейший класс, отнаследованный от *QDialog*, который содержит поля ширины и высоты, в которые записываются значения введенные пользователем. Окно приводится в действие путем использования метода *exec()* класса *QDialog*. Для завершения работы с окном реализовано два слота нажатия на кнопки «Отмена» и «ОК», *on_Cancel_clicked* и *on_Apply_clicked* соответственно. При вызове второго слота в поля класса записываются значения введенные пользователем. Затем окно закрывается для любого из слотов.

Внешний вид окна:



4.2. CreateFrameDialog

Класс отнаследованный от *QDialog*.

Содержит поля:

double radius — для хранения радиуса окружности узора «Крест»; *double angle* — для хранения угла поворота ветви узора «Ветви»; *int density* — определяет количество обводок узора «обод»; *int func* — отвечает за текущий отображаемый узор в предпросмотре; *Grapher *map* — предпросмотр рисунка.

Слоты:

on_apply_clicked() — нажатие на *«ОК»*; *on_reject_clicked()* — нажатие на *«Отмена»*

on_branches_clicked() - отобразить ветвь на предпросмотре

on_angle_value_valueChanged(int value) — отвечает за смену значения поля *angle*

on_line_frame_clicked() - отобразить обводящую рамку на предпросмотре

on_density_value_valueChanged(int value) — отвечает за смену значения поля *density*

on_kelt_cross_clicked() - отобразить крестовую рамку на предпросмотре

on_radius_value_valueChanged(int value) — отвечает за смену значения поля *radius*

Методы класса:

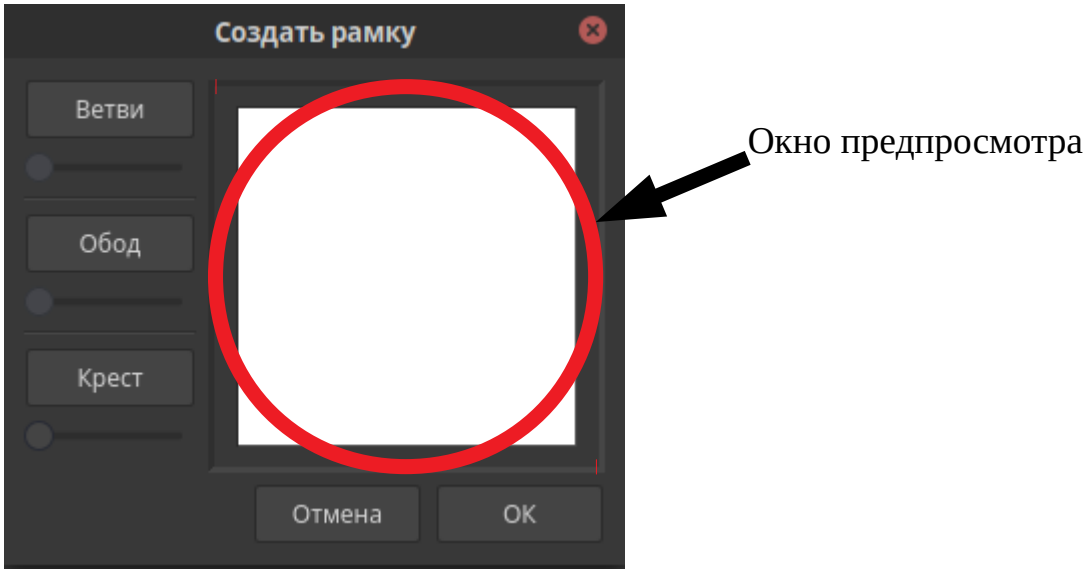
Все методы предназначены для отображения узора в предпросмотре.

drawBranch(double angle) — со сцены убираются все объекты, которые были нарисованы ранее. Создается точка начала отрисовки, затем циклом рисуются прямые начиная от конца предыдущей, пока длина последующй прямой не станет меньше 3. При этом на первой иттерации создается вторая ветвь направленная в другую сторону, с углом отклонения в два раза меньше и минимальной длиной равной 5.

drawLineFrame(int density) — в зависимости от значения *density* вычисляется толщина и количество линий путем использования оператора *switch*. Далее по каждой стороне отрисовывается полученное количество линий.

drawKeltCross(double radius) — рисуется крест и окружность с центром в середине окна предпросмотра. Радиус окружности определяется значением *radius*.

Внешний вид окна:

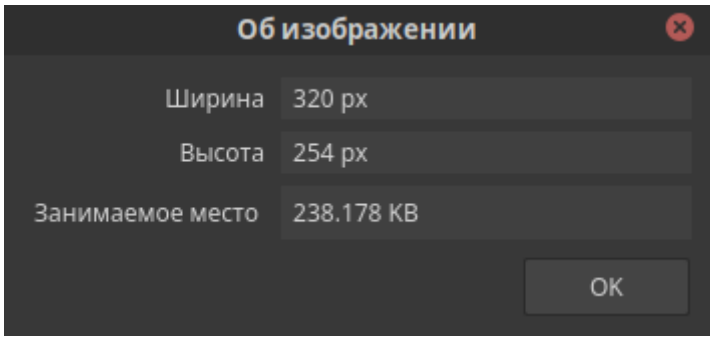


4.3. ImageInfo

Класс наследуется от *QDialog*.

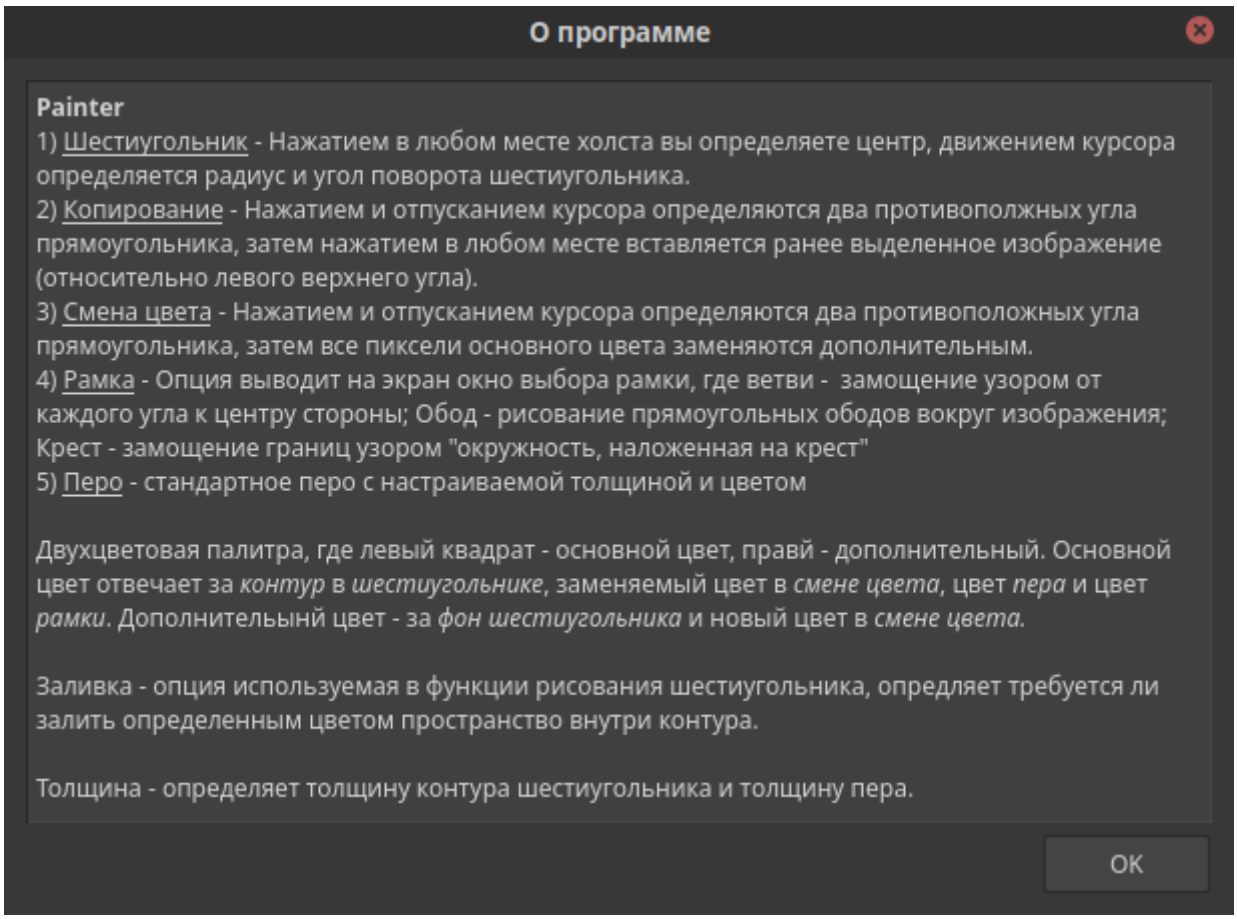
Класс содержит только один созданный метод — *update(Image *img)*, который просто располагает элементах окна требуемые значения. В ширину, высоту и занимаемое место, *bih.biWidth*, *bih.biHeight* и *bfh.bfSize* соответственно.

Внешний вид окна:



4.4. About

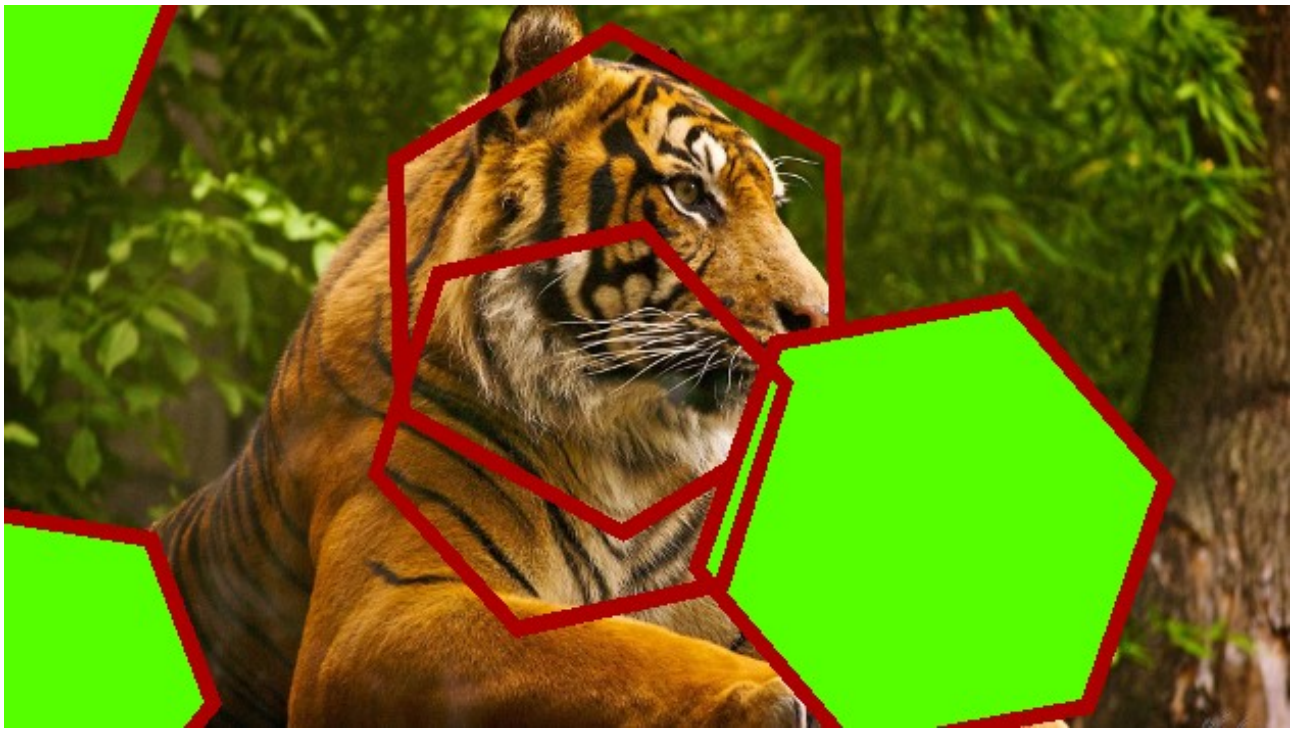
Класс наследуется от *QDialog*. Не содержит никаких полей помимо стандартных. Демонстрирует правила пользования программой.



5. ТЕСТИРОВАНИЕ

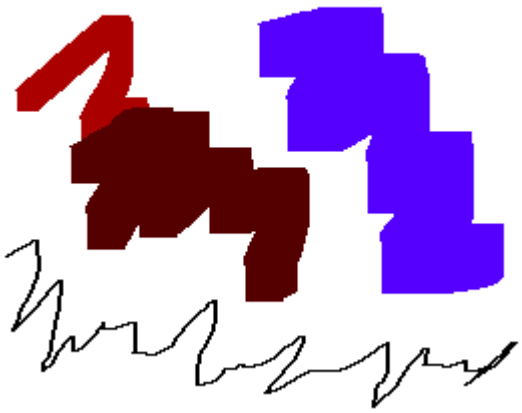
Программа будет тестироваться на следующем изображении:

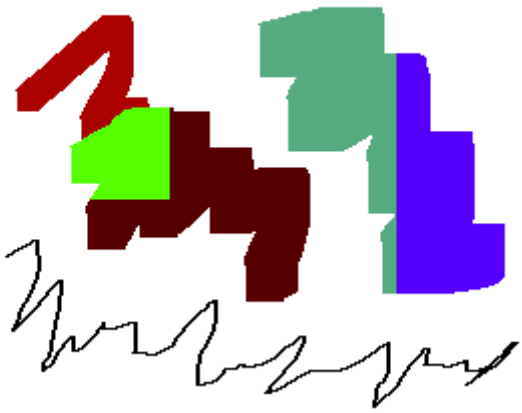
5.1. Тестирование функции рисования шестиугольника



5.2. Тестирование копирования

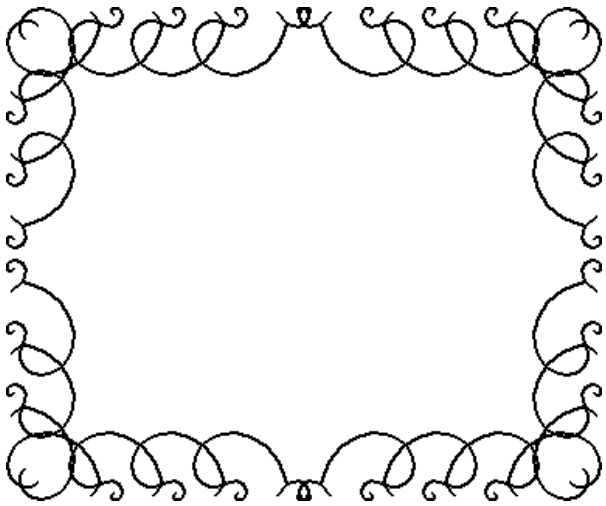
5.3. Создание нового изображения, использование пера и смены цвета



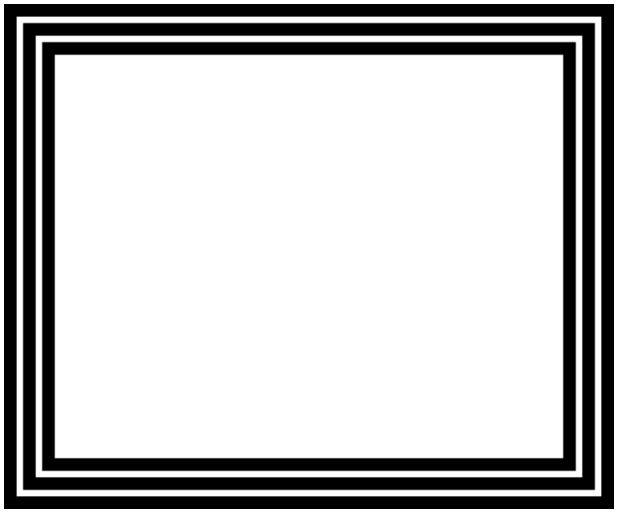


5.4. Тестирование рамки

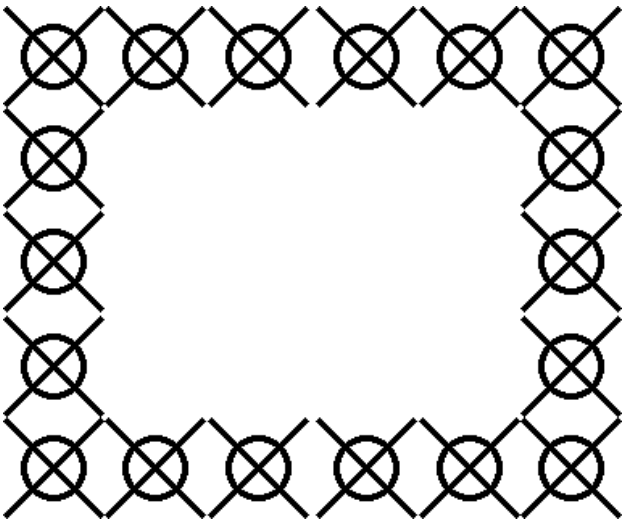
Ветви



Обод



Крест



ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была написана программа для обработки растрового изображения формата BMP на языке C++ с использованием графического фреймворка Qt. Для написания программы использовалась среда обработки Qt Creator и редактор форм Qt Designer. Была изучена структура BMP файла, а также различные стандартные классы Qt.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация фреймворка Qt. http://doc.crossplatform.ru/qt/

2. Документация C++. http://www.cplusplus.com/reference/

3. Форум для решения проблем связанных с Qt, а также поиск алгоритмов. https://stackoverflow.com/questions/tagged/qt

4. Жасмин Бланшетт, Марк Саммерфилд «Qt4 Программирование GUI на C++».: изд-во «КУДИЦ-Пресс». 2008 г. - 718 стр.

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

main.cpp

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QFileDialog>
#include <QColorDialog>
#include <QPalette>
#include <QMessageBox>
#include <string>
#include "image.h"
#include "newimagedialog.h"
#include "grapher.h"
#include "createframedialog.h"
#include "imageinfo.h"
#include "about.h"


namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_new_file_triggered();
    void on_save_file_triggered();
    void on_open_file_triggered();
    void on_about_image_triggered();
    void on_about_triggered();

    void on_Fill_stateChanged(int arg1);
    void on_Width_valueChanged(int arg1);
    void on_foreground_clicked();
    void on_background_clicked();
    void show_coordinates(int x, int y);

    void on_Hexagon_clicked();
    void on_Copy_clicked();
    void on_Recolor_clicked();
    void on_Pen_clicked();
```

```
    void pen();
    void hexagon(int sx, int sy, double radius, double angle);
    void copy(int sx, int sy, int fx, int fy, int nx, int ny);
    void recolor(int sx, int sy, int fx, int fy);
    void on_Frame_clicked();


private:
    QColor fore = QColor(0, 0, 0);
    QColor back = QColor(255, 255, 255);
    Image *img;                 // изображение
    Grapher *grapher;           // видимые операции
    bool fill = false;          // заливка
    int thick = 10;              // толщина линий
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include <QDebug>
#include "mainwindow.h"
#include "ui_mainwindow.h"


MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    grapher = new Grapher;
    grapher->scene->setSceneRect(0, 0, 512, 512);
    img = new Image;
    ui->setupUi(this);
    ui->Field->addWidget(grapher);

    QPalette forecolor = ui->foreground->palette();                 // Изъятие
палитры из кнопки осн. цвета
    QPalette backcolor = ui->background->palette();                 // Изъятие
палитры из кнопки доп. цвета
    forecolor.setColor(QPalette::Button, QColor(0, 0, 0));          // Изменение
палитры кнопки осн. цвета (на черный)
    backcolor.setColor(QPalette::Button, QColor(255, 255, 255));    // Изменение
палитры кнопки доп. цвета (на белый )
    ui->foreground->setPalette(forecolor);                          //
Применение измененной палитры к кнопке осн. цвета
    ui->background->setPalette(backcolor);                          //
Применение измененной палитры к кнопке доп. цвета
    ui->foreground->update();                                       //
Обновление кнопки осн. цвета
    ui->background->update();                                       //
Обновление кнопки доп. цвета

    connect(grapher, SIGNAL(copy_signal(int, int, int, int, int, int)), this,
SLOT(copy(int, int, int, int, int, int)));
    connect(grapher, SIGNAL(square_signal(int, int, int, int)), this,
SLOT(recolor(int, int, int, int)));
    connect(grapher, SIGNAL(hexagon_signal(int, int, double, double)),
SLOT(hexagon(int, int, double, double)));

    connect(grapher, SIGNAL(pen_signal()), this, SLOT(pen())); // Соединение
сигнала с мыши (при условии выбранной функции) для вызова пера
```

```
    connect(grapher, SIGNAL(mouse_track_signal(int, int)), this,
SLOT(show_coordinates(int, int))); // Соединение сигнала с мыши для вывода
координат в статусбар
}

MainWindow::~MainWindow()
{
    delete grapher;
    delete img;
    delete ui;
}


void MainWindow::on_about_triggered()
{
    About *abt = new About;
    abt->exec();
    delete abt;
    return;
}


void MainWindow::on_about_image_triggered()
{
    if(img->bih.biWidth == 0 || img->bih.biHeight == 0){
        QMessageBox::warning(this, "Ошибка", "Отсутствует изображение");
        return;
    }
    ImageInfo *info = new ImageInfo;
    info->update(img);
    info->exec();
    delete info;
    return;
}


void MainWindow::on_new_file_triggered()
{
    NewImageDialog *imageSize = new NewImageDialog;             // Вызов окна
создания нового изображения
    imageSize->exec();
    if (imageSize->result()){
        img->newImage(imageSize->width, imageSize->height);
        grapher->update(img->getMap());
    }
    grapher->scene->setSceneRect(0, 0, imageSize->width, imageSize->height);
    delete imageSize;
    return;
}


void MainWindow::on_save_file_triggered()
{
    QString path = QFileDialog::getSaveFileName(nullptr, "Сохранение", "/home/",
"*.bmp");
    path += ".bmp";
    int error = img->saveImage(path.toLocal8Bit().data());
    if(error){
        QMessageBox::critical(this, "Ошибка", "Неизвестная ошибка при сохранении
файла");
        return;
    }
}
```

```
void MainWindow::on_open_file_triggered()
{
    QString path = QFileDialog::getOpenFileName(nullptr, "Загрузка", "/home/",
"*.bmp");
    if(path == nullptr) return;
    int error = img->loadImage(path.toLocal8Bit().data());
    if (error == 1){
        QMessageBox::critical(this, "Ошибка", "Неизвестная ошибка при чтении
файла");
        return;
    }
    if (error == 2){
        QMessageBox::critical(this, "Ошибка", "Файл не является BMP
изображением");
        return;
    }
    if (error == 3){
        QMessageBox::warning(this, "Ошибка", "Размер изображения должен быть не
больше 8192x6144px");
        return;
    }
    if (error == 4){
        QMessageBox::critical(this, "Ошибка", "Программа не поддерживает
изображения с таблицей цветов");
        return;
    }
    grapher->scene->setSceneRect(0, 0, img->bih.biWidth, img->bih.biHeight);
    grapher->update(img->getMap());
}


void MainWindow::on_Fill_stateChanged(int arg1)
{
    if (arg1) fill = true;
    else fill = false;
}


void MainWindow::on_Width_valueChanged(int arg1)
{
    thick = arg1;
}


void MainWindow::on_foreground_clicked()    // Изменение основного цвета в
рантайме (аналогично конструктору)
{
    fore = QColorDialog::getColor();
    if(!fore.isValid()) return;
    QPalette forecolor = ui->foreground->palette();
    forecolor.setColor(QPalette::Button, fore);
    ui->foreground->setPalette(forecolor);
    ui->foreground->update();
}


void MainWindow::on_background_clicked()    // Изменение дополнительного цвета в
рантайме (аналогично конструктору)
{
    back = QColorDialog::getColor();
    if(!back.isValid()) return;
    QPalette backcolor = ui->background->palette();
    backcolor.setColor(QPalette::Button, back);
```

```
    ui->background->setPalette(backcolor);
    ui->background->update();
}


void MainWindow::on_Hexagon_clicked()
{
    if(img->bih.biWidth == 0 || img->bih.biHeight == 0){
        QMessageBox::warning(this, "Ошибка", "Отсутствует изображение");
        return;
    }
    grapher->copy = 0;
    grapher->func = HEXAGON;
}


void MainWindow::on_Copy_clicked()
{
    if(img->bih.biWidth == 0 || img->bih.biHeight == 0){
        QMessageBox::warning(this, "Ошибка", "Отсутствует изображение");
        return;
    }
    grapher->copy = 0;
    grapher->func = COPY;
}


void MainWindow::on_Recolor_clicked()
{
    if(img->bih.biWidth == 0 || img->bih.biHeight == 0){
        QMessageBox::warning(this, "Ошибка", "Отсутствует изображение");
        return;
    }
    grapher->copy = 0;
    grapher->func = RECOLOR;
}



void MainWindow::on_Frame_clicked()
{
    if(img->bih.biWidth == 0 || img->bih.biHeight == 0){
        QMessageBox::warning(this, "Ошибка", "Отсутствует изображение");
        return;
    }
    grapher->copy = 0;
    if(img->bih.biWidth < 50 || img->bih.biHeight < 50){
        QMessageBox::warning(this, "Ошибка", "Каждая сторона изображения должна
быть больше 50px");
        return;
    }

    CreateFrameDialog *frameParameters = new CreateFrameDialog;
    frameParameters->setFixedSize(312, 253);
    frameParameters->exec();

    int error = 0;
    if (frameParameters->result()){
        switch (frameParameters->func){
        case BRANCH:
            error = img->createFrame(fore, back, fill, BRANCH, frameParameters-
>angle);
            break;
```

```
        case RIM:
            error = img->createFrame(fore, back, fill, RIM, frameParameters-
>density);
            break;

        case CROSS:
            error = img->createFrame(fore, back, fill, CROSS, (frameParameters-
>radius*200)/SQR_SIDE);
            break;
        }
    }
    delete frameParameters;
    if(error){
        qDebug() << "error";
        return;
    }
    grapher->update(img->getMap());
    return;
}


void MainWindow::on_Pen_clicked()
{
    if (img->bih.biWidth == 0 || img->bih.biHeight == 0){
        QMessageBox::warning(this, "Ошибка", "Отсутствует изображение");
        return;
    }
    grapher->copy = 0;
    grapher->func = PEN;
}


void MainWindow::show_coordinates(int x, int y){    // Конструирование строк по
значениям сообщенным в слот и их вывод
    std::string str_x = std::to_string(x);
    std::string str_y = std::to_string(y);
    ui->statusBar->showMessage(QString::fromStdString("x = " + str_x + "; " + "y
= " + str_y + ";"));
}


void MainWindow::pen(){ // Рисует мелкие отрезки пока зажата мышь
    int error = img->setPart(grapher->start.x, grapher->start.y, grapher-
>finish.x, grapher->finish.y, fore, thick);
    if(error){
        qDebug() << "error";
        return;
    }
    grapher->update(img->getMap());
}


void MainWindow::hexagon(int sx, int sy, double radius, double angle){
    int error = img->paintHexagon(sx, sy, radius, angle, thick, fill, fore,
back);
    if(error){
        qDebug() << "error";
        return;
    }
    grapher->update(img->getMap());
}


void MainWindow::copy(int sx, int sy, int fx, int fy, int nx, int ny){
```

```
    int error = img->copyRect(sx, sy, fx, fy, nx, ny);
    if (error){
        qDebug() << "position error";
        return;
    }
    grapher->update(img->getMap());
}


void MainWindow::recolor(int sx, int sy, int fx, int fy){
    int error = img->recolorRect(sx, sy, fx, fy, fore, back);
    if (error){
        qDebug() << "error";
        return;
    }
    grapher->update(img->getMap());
}
```

grapher.h

```
#ifndef GRAPHER_H
#define GRAPHER_H

#include <QGraphicsView>
#include <QGraphicsScene>
#include <QGraphicsItemGroup>
#include <QMouseEvent>
#include <QPoint>
#include <cmath>

#define PI 3.14159265359

#define OFF 0
#define PEN 1
#define COPY 2
#define RECOLOR 3
#define HEXAGON 4

typedef struct coordinate{
    int x;
    int y;
} coordinate;

class Grapher : public QGraphicsView
{

    Q_OBJECT

public:
    QGraphicsScene *scene;
    coordinate start;
    coordinate finish;
    int func = OFF;
    int copy = 0;
    void update(QPixmap);
    explicit Grapher(QWidget *Parent = nullptr);
    ~Grapher();

signals:
    void mouse_track_signal(int x, int y);
    void pen_signal();
    void square_signal(int sx, int sy, int fx, int fy);
    void copy_signal(int sx, int sy, int fx, int fy, int nx, int ny);
    void hexagon_signal(int sx, int sy, double radius, double angle);
```

```
private slots:
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *event);
    void mouseReleaseEvent(QMouseEvent *event);

private:
    QGraphicsItemGroup *object;

    QGraphicsItemGroup *hexagon;
    double radius;
    double angle;

    QPixmap pixmap;
    bool drawer = false;
    void freeGroup(QGraphicsItemGroup *object);
};

#endif // GRAPHER_H
```

grapher.cpp

```
#include <QDebug>
#include "grapher.h"

Grapher::Grapher(QWidget *parent):QGraphicsView(parent)
{
    this->setAlignment(Qt::AlignLeft | Qt::AlignTop);
    scene = new QGraphicsScene();
    this->setScene(scene);

    object = new QGraphicsItemGroup();
    hexagon = new QGraphicsItemGroup();
    scene->addItem(object);
    scene->addItem(hexagon);

    setMouseTracking(true);
}
Grapher::~Grapher(){}


void Grapher::update(QPixmap pixmap){
    foreach(QGraphicsItem *item, scene->items())
        if(typeid(*item) == typeid(QGraphicsPixmapItem))
            scene->removeItem(item);

    scene->addPixmap(pixmap);
}


void Grapher::freeGroup(QGraphicsItemGroup* object){
    foreach(QGraphicsItem *item, scene->items())
        if (item->group() == object)
            delete item;
}


void Grapher::mousePressEvent(QMouseEvent *event){
    radius = 0;
    angle = 0;
    QPoint point = mapToScene(event->pos()).toPoint();
    emit mouse_track_signal(point.x(), point.y());
    drawer = true;

    if (copy){
        copy = 0;
```

```
        emit copy_signal(start.x, start.y, finish.x, finish.y, point.x(),
point.y());
        drawer = false;
    }

    start.x = point.x();
    start.y = point.y();
    finish.x = point.x();
    finish.y = point.y();
}


void Grapher::mouseMoveEvent(QMouseEvent *event){
    QPoint point = mapToScene(event->pos()).toPoint();
    emit mouse_track_signal(point.x(), point.y());

    if (drawer == true && func == PEN){
        finish.y = point.y();
        finish.x = point.x();
        emit pen_signal();
        start.x = finish.x;
        start.y = finish.y;
    }
    else if (drawer == true && (func == COPY || func == RECOLOR)){
        finish.y = point.y();
        finish.x = point.x();

        this->freeGroup(object);
        delete object;
        object = new QGraphicsItemGroup;

        QPen pen(Qt::black);
        if(finish.x < 0) finish.x = 0;
        if(finish.y < 0) finish.y = 0;

        object->addToGroup(scene->addLine(start.x, start.y, finish.x, start.y,
pen));
        object->addToGroup(scene->addLine(finish.x, start.y, finish.x, finish.y,
pen));
        object->addToGroup(scene->addLine(finish.x, finish.y, start.x, finish.y,
pen));
        object->addToGroup(scene->addLine(start.x, finish.y, start.x, start.y,
pen));

        scene->addItem(object);
    }
    else if (drawer == true && func == HEXAGON){
        finish.x = point.x();
        finish.y = point.y();

        this->freeGroup(hexagon);
        delete hexagon;
        hexagon = new QGraphicsItemGroup;

        QPen pen(Qt::black);
        radius = sqrt((finish.x - start.x)*(finish.x - start.x) + (finish.y -
start.y)*(finish.y - start.y));
        angle = acos((finish.x - start.x)/radius);
        if (finish.y < start.y) angle *= -1;

        int s_node_x = finish.x;
        int s_node_y = finish.y;
        int f_node_x;
        int f_node_y;
```

```
        for(int i = 1; i <= 6; i++){
            f_node_x = static_cast<int>(start.x + round(radius*cos(PI/3*i +
angle)));
            f_node_y = static_cast<int>(start.y + round(radius*sin(PI/3*i +
angle)));
            hexagon->addToGroup(scene->addLine(s_node_x, s_node_y, f_node_x,
f_node_y, pen));
            s_node_x = f_node_x;
            s_node_y = f_node_y;
        }
        scene->addItem(hexagon);
    }
}


void Grapher::mouseReleaseEvent(QMouseEvent *event){
    if(func == COPY && finish.x - start.x && finish.y - start.y) copy = 1;

    QPoint point = mapToScene(event->pos()).toPoint();
    finish.x = point.x();
    finish.y = point.y();

    this->freeGroup(object);
    this->freeGroup(hexagon);
    drawer = false;

    if(func == PEN) emit pen_signal();
    if(func == RECOLOR) emit square_signal(start.x, start.y, finish.x,
finish.y);
    if(func == HEXAGON) emit hexagon_signal(start.x, start.y, radius, angle);
}
```

image.h

```
#ifndef IMAGE_H
#define IMAGE_H

#include <QPixmap>
#include <QMessageBox>
#include <cmath>
#include <fstream>
#include <stdint.h>
#include <stack>
#include <algorithm>
#include "grapher.h"

#define PI 3.14159265359
#define VERTICAL 1
#define HORIZONTAL 0

class Image
{

#pragma pack(push, 1)
    typedef struct{
        uint16_t bfType         = 0;
        uint32_t bfSize         = 0;
        uint16_t bfReserved1    = 0;
        uint16_t bfReserved2    = 0;
        uint32_t bfOffBits      = 0;
    }BITMAPFILEHEADER;

    typedef struct{
        uint32_t biSize         = 0;
        int32_t biWidth         = 0;
```

```
        int32_t biHeight        = 0;
        uint16_t biPlanes       = 0;
        uint16_t biBitCount     = 0;
        uint32_t biCompression  = 0;
        uint32_t biSizeImage    = 0;
        int32_t biXPelsPerMeter = 0;
        int32_t biYPelsPerMeter = 0;
        uint32_t biClrUsed      = 0;
        uint32_t biClrImportant = 0;
    }BITMAPINFOHEADER;
#pragma pack(pop)

    typedef struct RGB{
        unsigned char red;
        unsigned char green;
        unsigned char blue;
        unsigned char visible = 1;
        friend bool operator!=(const RGB& left, const RGB& right){
            return !(left.red == right.red &&
                    left.green == right.green &&
                    left.blue == right.blue &&
                    left.visible == right.visible);
        }
    }RGB;

public:
    ~Image();
    BITMAPFILEHEADER bfh;
    BITMAPINFOHEADER bih;
    RGB** rgb = nullptr;
    QPixmap getMap();

    void floodFill(int x, int y, QColor color, bool fill);
    int setPart(int s_x, int s_y, int f_x, int f_y, QColor color, int thick);
    void drawLine(int s_x, int s_y, int f_x, int f_y, QColor color, bool fill =
1);
    void drawThickLine(int s_x, int s_y, int f_x, int f_y, int thick, QColor
color, bool fill = 1);
    void defCircle(int s_x, int s_y, double radius, QColor color, bool fill =
1);
    void drawCircle(int s_x, int s_y, double radius, int thick, QColor color);
    void mirror_hor();
    void mirror_ver();
    void rotate();

    int defHexagon(int s_x, int s_y, double radius, double angle, QColor color,
bool fill = 1);
    int paintHexagon(int s_x, int s_y, double radius, double angle, int thick,
bool fill, QColor fore, QColor back);
    int copyRect(int s_x, int s_y, int f_x, int f_y, int n_x, int n_y);
    int recolorRect(int s_x, int s_y, int f_x, int f_y, QColor old_color, QColor
new_color);
    int createFrame(QColor primary, QColor additional, bool fill, int type,
double parameter);

    int newImage(const int32_t width, const int32_t height);
    int saveImage(const char *path);
    int loadImage(const char *path);
};

#endif // IMAGE_H
```

image.cpp

```
#include "image.h"
#include <QDebug>



int Image::newImage(const int32_t width, const int32_t height){

        /*
         * Настройки первых двух
         * Блоков памяти нового BMP файла
         */
    bfh.bfType = 0x4d42;
    bfh.bfSize = (unsigned(width)*3 + (4 -
(unsigned(width)*3)%4)%4)*unsigned(height) + 54;
    bfh.bfReserved1 = 0;
    bfh.bfReserved2 = 0;
    bfh.bfOffBits = 54;

    bih.biSize = 40;
    bih.biWidth = width;
    bih.biHeight = height;
    bih.biPlanes = 1;
    bih.biClrUsed = 0;
    bih.biBitCount = 24;
    bih.biSizeImage = unsigned(width*height);
    bih.biCompression = 0;
    bih.biClrImportant = 0;
    bih.biXPelsPerMeter = 0;
    bih.biYPelsPerMeter = 0;

    /* Создание матрицы пикселей */
    rgb = new RGB* [unsigned(height)];
    for(int i = 0; i < height; i++){
        rgb[i] = new RGB [unsigned(width)];
        for(int j = 0; j < width; j++){
            rgb[i][j] = {255, 255, 255};
        }
    }

//    drawThickLine(10, 10, 50, 50, 10, Qt::black);
    return 0;
}


int Image::saveImage(const char *path){
    FILE *out = fopen(path, "wb");
    if (!out) return 1;
    fwrite(&bfh, sizeof(bfh), 1, out);
    fwrite(&bih, sizeof(bih), 1, out);
    fseek(out, long(bfh.bfOffBits), SEEK_SET);

    size_t len_of_pdng = (4 - (bih.biWidth*3)%4)%4;             // Расчет длины
выравнивания
    char *pdng = static_cast<char*>(new char[len_of_pdng]);     // Создание
массива нулей для выравнивания

    for(int i = bih.biHeight - 1; i >= 0; i--){                 //
Инвертирование в высоту в силу расположения битов в BMP
        for(int j = 0; j < bih.biWidth; j++){
//            fwrite(&(rgb[i][j]), sizeof(RGB), 1, out);
            fwrite(&(rgb[i][j].blue), sizeof(uchar), 1, out);
            fwrite(&(rgb[i][j].green), sizeof(uchar), 1, out);
```

```
            fwrite(&(rgb[i][j].red), sizeof(uchar), 1, out);
        }
        fwrite(pdng, len_of_pdng, 1, out);
    }
    fclose(out);
    delete [] pdng;
    return 0;
}


int Image::loadImage(const char *path){
    FILE *inp = fopen(path, "rb");
    if (!inp) return 1;

    for(int i = 0; i < bih.biHeight; i++)
        delete [] rgb[i];
    delete [] rgb;

    fread(&bfh, sizeof(BITMAPFILEHEADER), 1, inp);

    if(bfh.bfType != 0x4d42){
        bfh = {0, 0, 0, 0, 0};
        fclose(inp);
        return 2;
    }

    fread(&bih, sizeof(BITMAPINFOHEADER), 1, inp);
    if (bih.biWidth > 8192 || bih.biHeight > 6144){
        bfh = {0, 0, 0, 0, 0};
        bih = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        fclose(inp);
        return 3;
    }
    if (bih.biClrUsed){
        bfh = {0, 0, 0, 0, 0};
        bih = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        fclose(inp);
        return 4;
    }

    rgb = new RGB* [unsigned(bih.biHeight)];
    for(int i = 0; i < bih.biHeight; i++)
        rgb[i] = new RGB [unsigned(bih.biWidth)];

    fseek(inp, long(bfh.bfOffBits), SEEK_SET);
    long int len_of_pdng = (4 - (bih.biWidth*3)%4)%4;

    for(int i = bih.biHeight - 1; i >= 0; i--){
        for(int j = 0; j < bih.biWidth; j++){
            fread(&(rgb[i][j].blue), sizeof(unsigned char), 1, inp);
            fread(&(rgb[i][j].green), sizeof(unsigned char), 1, inp);
            fread(&(rgb[i][j].red), sizeof(unsigned char), 1, inp);
            rgb[i][j].visible = 1;
        }
        fseek(inp, len_of_pdng, SEEK_CUR);
    }
    fclose(inp);
    return 0;
}


Image::~Image(){
    for(int i = 0; i < bih.biHeight; i++)
        delete [] rgb[i];
```

```
    delete [] rgb;
}


QPixmap Image::getMap(){
    /* Построение изображения с помощью матрицы пикселей rgb */
    QImage *image = new QImage(bih.biWidth, bih.biHeight,
QImage::Format_RGB666);
    QColor pixel;
    for(int i = 0; i < bih.biHeight; i++){
        for(int j = 0; j < bih.biWidth; j++){
            pixel.setRed(rgb[i][j].red);
            pixel.setGreen(rgb[i][j].green);
            pixel.setBlue(rgb[i][j].blue);
            image->setPixel(j,i,pixel.rgb());
        }
    }
    QPixmap out = QPixmap::fromImage(*image);
    delete image;
    return out;
}



typedef struct{
    int x;
    int y;
}point;


void Image::floodFill(int x, int y, QColor color, bool fill = 1){
    point pt, curr;
    RGB pix = {uchar(color.red()), uchar(color.green()), uchar(color.blue()),
fill};

    pt.x = x;
    pt.y = y;
    std::stack<point> stk;
    rgb[pt.y][pt.x] = {uchar(color.red()), uchar(color.green()),
uchar(color.blue()), fill};
    stk.push(pt);

    while(!stk.empty()){
        curr = stk.top();
        stk.pop();

        if (curr.x > 0 && rgb[curr.y][curr.x - 1] != pix){
            pt.x = curr.x - 1;
            pt.y = curr.y;
            rgb[pt.y][pt.x] = {uchar(color.red()), uchar(color.green()),
uchar(color.blue()), fill};
            stk.push(pt);
        }

        if (curr.x + 1 < bih.biWidth && rgb[curr.y][curr.x + 1] != pix){
            pt.x = curr.x + 1;
            pt.y = curr.y;
            rgb[pt.y][pt.x] = {uchar(color.red()), uchar(color.green()),
uchar(color.blue()), fill};
            stk.push(pt);
        }

        if (curr.y > 0 && rgb[curr.y - 1][curr.x] != pix){
            pt.x = curr.x;
```

```
            pt.y = curr.y - 1;
            rgb[pt.y][pt.x] = {uchar(color.red()), uchar(color.green()),
uchar(color.blue()), fill};
            stk.push(pt);
        }

        if (curr.y + 1 < bih.biHeight && rgb[curr.y + 1][curr.x] != pix){
            pt.x = curr.x;
            pt.y = curr.y + 1;
            rgb[pt.y][pt.x] = {uchar(color.red()), uchar(color.green()),
uchar(color.blue()), fill};
            stk.push(pt);
        }
    }
}


int Image::setPart(int s_x, int s_y, int f_x, int f_y, QColor color, int thick){
    for(int s_i = s_x - thick/2, f_i = f_x - thick/2; s_i <= s_x + thick/2 ||
f_i <= f_x + thick/2; s_i++, f_i++){
        for(int s_j = s_y - thick/2, f_j = f_y - thick/2; s_j <= s_y + thick/2
|| f_j <= f_y + thick/2; s_j++, f_j++){
            if (    (s_j >= 0) && (s_j < bih.biHeight) &&
                    (s_i >= 0) && (s_i < bih.biWidth)  &&
                    (f_j >= 0) && (f_j < bih.biHeight) &&
                    (f_i >= 0) && (f_i < bih.biWidth)     ){
                rgb[s_j][s_i].red = uchar(color.red());
                rgb[s_j][s_i].green = uchar(color.green());
                rgb[s_j][s_i].blue = uchar(color.blue());
                rgb[f_j][f_i].red = uchar(color.red());
                rgb[f_j][f_i].green = uchar(color.green());
                rgb[f_j][f_i].blue = uchar(color.blue());
            }
        }
    }

    if ((s_x - f_x) * (s_y - f_y) > 0){
        for(int i = -thick/2; i <= thick/2; i++){
            drawLine(s_x - i, s_y + i, f_x - i, f_y + i, color);
            drawLine(s_x - i - 1, s_y + i, f_x - i - 1, f_y + i, color);
        }
    }
    else {
        for(int i = -thick/2; i <= thick/2; i++){
            drawLine(s_x + i, s_y + i, f_x + i, f_y + i, color);
            drawLine(s_x + i - 1, s_y + i, f_x + i - 1, f_y + i, color);
        }
    }
    return 0;
}


void Image::drawLine(int s_x, int s_y, int f_x, int f_y, QColor color, bool
fill){
    const int deltaX = abs(f_x - s_x);
    const int deltaY = abs(f_y - s_y);
    const int signX = s_x < f_x ? 1 : -1;
    const int signY = s_y < f_y ? 1 : -1;

    int error = deltaX - deltaY;

    if(f_y >= 0 && f_x >= 0 && f_y < bih.biHeight && f_x < bih.biWidth){
        rgb[f_y][f_x].red = uchar(color.red());
        rgb[f_y][f_x].green = uchar(color.green());
```

```
        rgb[f_y][f_x].blue = uchar(color.blue());
        rgb[f_y][f_x].visible = fill;
    }

    while(s_x != f_x || s_y != f_y){
        if(s_y >= 0 && s_x >= 0 && s_y < bih.biHeight && s_x < bih.biWidth){
            rgb[s_y][s_x].red = uchar(color.red());
            rgb[s_y][s_x].green = uchar(color.green());
            rgb[s_y][s_x].blue = uchar(color.blue());
            rgb[s_y][s_x].visible = fill;
        }

        const int error2 = error*2;

        if(error2 > -deltaY){
            error -= deltaY;
            s_x += signX;
        }
        if(error2 < deltaX){
            error += deltaX;
            s_y += signY;
        }
    }
}


void Image::drawThickLine(int s_x, int s_y, int f_x, int f_y, int thick, QColor
color, bool fill){
    if (thick == 1){
        drawLine(s_x, s_y, f_x, f_y, color, fill);
        return;
    }

       // VERTICAL
    for(int i = 0; i < thick; i++){
        drawLine(s_x, s_y - i/2, f_x, f_y - i/2, color, fill);
    }
       // HORIZONTAL
    for(int i = 0; i < thick; i++){
        drawLine(s_x - i/2, s_y, f_x - i/2, f_y, color, fill);
    }
}


void Image::defCircle(int s_x, int s_y, double radius, QColor color, bool fill){
    for(int i = -int(radius); i <= int(radius); i++){
        for(int j = -int(radius); j <= int(radius); j++){
            if(i*i + j*j <= int(radius*radius))
                rgb[s_y + i][s_x + j] = {uchar(color.red()),
uchar(color.green()), uchar(color.blue()), fill};
        }
    }
}


void Image::drawCircle(int s_x, int s_y, double radius, int thick, QColor color)
{
    int32_t bufSide = static_cast<int32_t>(round((radius+1)*2));
    Image *buffer =  new Image;
    buffer->bih.biHeight = bufSide;
    buffer->bih.biWidth = bufSide;
    buffer->rgb = new RGB* [unsigned(bufSide)];
    for(int i = 0; i < bufSide; i++){
        buffer->rgb[i] = new RGB [unsigned(bufSide)];
```

```
        for(int j = 0; j < bufSide; j++)
            buffer->rgb[i][j].visible = 0;
    }
    int cntr = bufSide/2;

    buffer->defCircle(cntr, cntr, radius, color);
    if (thick) buffer->defCircle(cntr, cntr, radius - thick + 1, color, 0);

    for(int i = 0; i < bufSide; i++){
        for(int j = 0; j < bufSide; j++)
            if(s_y-cntr+i >= 0 && s_x-cntr+j >= 0 && s_y-cntr+i < bih.biHeight
&& s_x-cntr+j < bih.biWidth && buffer->rgb[i][j].visible)
                rgb[s_y-cntr+i][s_x-cntr+j] = buffer->rgb[i][j];
    }

    delete buffer;
    return;
}


void Image::mirror_hor(){
    for (int i = 0; i < bih.biHeight; i++)
        for(int j = 0; j <= bih.biWidth/2; j++){
            RGB buf = rgb[i][j];
            rgb[i][j] = rgb[i][bih.biWidth - 1 - j];
            rgb[i][bih.biWidth - 1 - j] =  buf;
        }
}


void Image::mirror_ver(){
    for (int i = 0; i <= bih.biHeight/2; i++)
        for(int j = 0; j < bih.biWidth; j++){
            RGB buf = rgb[i][j];
            rgb[i][j] = rgb[bih.biHeight - 1 - i][j];
            rgb[bih.biHeight - 1 - i][j] = buf;
        }
}


int Image::defHexagon(int s_x, int s_y, double radius, double angle, QColor
color, bool fill){
    int s_node_x = static_cast<int>(s_x + round(radius*cos(angle)));
    int s_node_y = static_cast<int>(s_y + round(radius*sin(angle)));
    int f_node_x;
    int f_node_y;
    for(int i = 0; i <= 6; i++){
        f_node_x = static_cast<int>(s_x + round(radius*cos(PI/3*i + angle)));
        f_node_y = static_cast<int>(s_y + round(radius*sin(PI/3*i + angle)));
        drawLine(s_node_x, s_node_y, f_node_x, f_node_y, color, fill);
        s_node_x = f_node_x;
        s_node_y = f_node_y;
    }
    this->floodFill(s_x, s_y, color, fill);
    return 0;
}


int Image::paintHexagon(int s_x, int s_y, double radius, double angle, int
thick, bool fill, QColor fore, QColor back){
    int32_t bufSide = static_cast<int32_t>(round((radius+1)*2));
    Image *buffer =  new Image;
    buffer->bih.biHeight = bufSide;
    buffer->bih.biWidth = bufSide;
```

```
    buffer->rgb = new RGB* [unsigned(bufSide)];
    for(int i = 0; i < bufSide; i++){
        buffer->rgb[i] = new RGB [unsigned(bufSide)];
        for(int j = 0; j < bufSide; j++)
            buffer->rgb[i][j] = {uchar(back.red()), uchar(back.green()),
uchar(back.blue()), 0};
    }
    int cntr = bufSide/2;
    buffer->defHexagon(cntr, cntr, radius, angle, fore);
    buffer->defHexagon(cntr, cntr, radius - thick + 1, angle, back, fill);

    for(int i = 0; i < bufSide; i++){
        for(int j = 0; j < bufSide; j++)
            if(s_y-cntr+i >= 0 && s_x-cntr+j >= 0 && s_y-cntr+i < bih.biHeight
&& s_x-cntr+j < bih.biWidth && buffer->rgb[i][j].visible)
                rgb[s_y-cntr+i][s_x-cntr+j] = buffer->rgb[i][j];
    }

    delete buffer;
    return 0;
}


int Image::copyRect(int s_x, int s_y, int f_x, int f_y, int n_x, int n_y){
    if(n_x <= 0 || n_y <= 0 || n_x > bih.biWidth || n_y > bih.biHeight) return
1;

    int ux;
    int uy;
    int lx;
    int ly;

    if (s_x < 0) s_x = 0;
    if (s_y < 0) s_y = 0;
    if (f_x < 0) f_x = 0;
    if (f_y < 0) f_y = 0;

    if (s_x >= bih.biWidth) s_x = bih.biWidth - 1;
    if (s_y >= bih.biHeight) s_y = bih.biHeight - 1;
    if (f_x >= bih.biWidth) f_x = bih.biWidth - 1;
    if (f_y >= bih.biHeight) f_y = bih.biHeight - 1;

    if (s_x > f_x) { ux = f_x; lx = s_x;}
    else {ux = s_x; lx = f_x;}

    if (s_y > f_y) {uy = f_y; ly = s_y;}
    else {uy = s_y; ly = f_y;}

    uint32_t bufHeight = static_cast<uint32_t>(ly - uy + 1);
    uint32_t bufWidth = static_cast<uint32_t>(lx - ux + 1);
    RGB **buffer = new RGB* [bufHeight];
    for(size_t i = 0; i < bufHeight; i++)
        buffer[i] = new RGB [bufWidth];

    for(int i = uy; i <= ly; i++)
        for(int j = ux; j <= lx; j++)
            buffer[i-uy][j-ux] = rgb[i][j];

    for(int i = n_y; i < n_y + int(bufHeight); i++)
        for(int j = n_x; j < n_x + int(bufWidth); j++)
            if(i >= 0 && j >= 0 && i < bih.biHeight && j < bih.biWidth)
                rgb[i][j] = buffer[i-n_y][j-n_x];

    for(size_t i = 0; i < bufHeight; i++)
```

```
        delete [] buffer[i];
    delete [] buffer;

    return 0;
}


int Image::recolorRect(int s_x, int s_y, int f_x, int f_y, QColor old_color,
QColor new_color){
    int ux;
    int uy;
    int lx;
    int ly;

    if (s_x < 0) s_x = 0;
    if (s_y < 0) s_y = 0;
    if (f_x < 0) f_x = 0;
    if (f_y < 0) f_y = 0;

    if (s_x >= bih.biWidth) s_x = bih.biWidth - 1;
    if (s_y >= bih.biHeight) s_y = bih.biHeight - 1;
    if (f_x >= bih.biWidth) f_x = bih.biWidth - 1;
    if (f_y >= bih.biHeight) f_y = bih.biHeight - 1;

    if (s_x > f_x) { ux = f_x; lx = s_x;}
    else {ux = s_x; lx = f_x;}

    if (s_y > f_y) {uy = f_y; ly = s_y;}
    else {uy = s_y; ly = f_y;}

    for(int i = uy; i <= ly; i++)
        for(int j = ux; j <= lx; j++){
            if(i >= 0 && j >= 0 && i < bih.biHeight && j < bih.biWidth)
                if( (rgb[i][j].red == uchar(old_color.red()))     &&
                    (rgb[i][j].green == uchar(old_color.green())) &&
                    (rgb[i][j].blue == uchar(old_color.blue()))        ){

                        rgb[i][j].red = uchar(new_color.red());
                        rgb[i][j].green = uchar(new_color.green());
                        rgb[i][j].blue = uchar(new_color.blue());
                }
        }

    return 0;
}


int Image::createFrame(QColor primary, QColor additional, bool fill, int type,
double parameter){
    Image *buffer = new Image;
    int elem_size = std::min(bih.biWidth, bih.biHeight)/5;

    buffer->bih.biWidth = bih.biWidth;
    buffer->bih.biHeight = bih.biHeight;
    buffer->rgb = new RGB* [unsigned(bih.biHeight)];
    for(int i = 0; i < bih.biHeight; i++){
        buffer->rgb[i] = new RGB [unsigned(bih.biWidth)];
        for(int j = 0; j < bih.biWidth; j++){
            buffer->rgb[i][j] = {uchar(primary.red()), uchar(primary.green()),
uchar(primary.blue()), 0};
        }
    }

    if (type == 2){                     // RIM
```

```
    int frame_count = 0;
    int width = 0;
    switch (int(parameter)){
    case 1:
        width = 16;
        frame_count = 1;
        break;
    case 2:
        width = 12;
        frame_count = 2;
        break;
    case 3:
        width = 8;
        frame_count = 3;
        break;
    case 4:
        width = 6;
        frame_count = 4;
        break;
    case 5:
        width = 4;
        frame_count = 5;
        break;
    }

    int s_x = 0;
    int s_y = 0;
    int f_x = bih.biWidth - 1;
    int f_y = bih.biHeight - 1;

    for(int i = 0; i < frame_count; i++){
        for(int y = s_y; y <= s_y + width - 1; y++)
            for(int x = s_x; x <= f_x; x++)
                buffer->rgb[y][x].visible = 1;

        for(int y = s_y + width - 1; y <= f_y - width + 1; y++)
            for(int x = s_x; x <= s_x + width - 1; x++)
                buffer->rgb[y][x].visible = 1;

        for(int y = f_y - width + 1; y <= f_y; y++)
            for(int x = s_x; x <= f_x; x++)
                buffer->rgb[y][x].visible = 1;

        for(int y = s_y + width - 1; y <= f_y - width + 1; y++)
            for(int x = f_x - width + 1; x <= f_x; x++)
                buffer->rgb[y][x].visible = 1;

        s_x += width*3/2;
        s_y += width*3/2;
        f_x -= width*3/2;
        f_y -= width*3/2;
    }
}
else if (type == 3 || type == 1){
    Image *elem = new Image;

    int width = 4;
    if (type == 3){                  // CROSS
        elem->bih.biWidth = elem_size;
        elem->bih.biHeight = elem_size;
        elem->rgb = new RGB* [unsigned(elem_size)];
        for(int i = 0; i < elem_size; i++){
            elem->rgb[i] = new RGB [unsigned(elem_size)];
            for(int j = 0; j < elem_size; j++)
```

```
                    elem->rgb[i][j] = {255, 255, 255, 1};
            }
            elem->drawCircle(2, 2, double(width)/2.0, 0, Qt::black);
            elem->drawCircle(elem_size-3, 2, double(width)/2.0, 0, Qt::black);
            elem->drawCircle(elem_size-3, elem_size-3, double(width)/2.0, 0,
Qt::black);
            elem->drawCircle(2, elem_size-3, double(width)/2.0, 0, Qt::black);

            for(int i = 0; i < 5; i++)
                elem->drawLine(2, 0 + i, elem_size-3, elem_size-5 + i,
Qt::black);
            for(int i = 0; i < 5; i++)
                elem->drawLine(elem_size-3, 0 + i, 2, elem_size-5 + i,
Qt::black);

            elem->drawCircle(elem_size/2, elem_size/2,
(double(parameter*elem_size))/200.0, 5, Qt::black);

            double pdng;
            int count;

            //horizontal
            pdng = (double(bih.biWidth %
elem_size))*double(elem_size)/double(bih.biWidth);
            count =  bih.biWidth/elem_size;
            for(int i = 0; i < count/2; i++){
                for(int y = 0; y < elem_size; y++)
                    for(int x = 0; x < elem_size; x++)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[y][x + int(i*(elem_size +
pdng))].visible = 1;
                            buffer->rgb[y+(bih.biHeight - (elem_size))][x +
int(i*(elem_size + pdng))].visible = 1;

                            buffer->rgb[y][bih.biWidth - ((i+1)*elem_size +
int(i*pdng)) + x].visible = 1;
                            buffer->rgb[y+(bih.biHeight - (elem_size))]
[bih.biWidth - ((i+1)*elem_size + int(i*pdng)) + x].visible = 1;
                        }
            }
            if (count%2)
                for(int y = 0; y < elem_size; y++)
                    for(int x = 0; x < elem_size; x++)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[y][(bih.biWidth/2 - 1) - elem_size/2 +
x].visible = 1;
                            buffer->rgb[y+(bih.biHeight - (elem_size))]
[(bih.biWidth/2 - 1) - elem_size/2 + x].visible = 1;
                        }

            //vertical
            pdng = (double(bih.biHeight %
elem_size))*double(elem_size)/double(bih.biHeight);
            count = bih.biHeight/elem_size;
            for(int i = 1; i < count/2; i++){
                for(int y = 0; y < elem_size; y++)
                    for(int x = 0; x < elem_size; x++)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[y + int(i*(elem_size + pdng))]
[x].visible = 1;
```

```
                            buffer->rgb[y + int(i*(elem_size + pdng))][x +
(bih.biWidth - elem_size)].visible = 1;

                            buffer->rgb[bih.biHeight - (i+1)*(elem_size +
int(pdng)) + y][x].visible = 1;
                            buffer->rgb[bih.biHeight - (i+1)*(elem_size +
int(pdng)) + y][x + (bih.biWidth - elem_size)].visible = 1;

                        }
            }
            if (count%2)
                for(int y = 0; y < elem_size; y++)
                    for(int x = 0; x < elem_size; x++)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[(bih.biHeight/2 - 1) - elem_size/2 + y]
[x].visible = 1;
                            buffer->rgb[(bih.biHeight/2 - 1) - elem_size/2 + y]
[x + (bih.biWidth - elem_size)].visible = 1;
                        }
        }

        if (type == 1){                 // BRANCH
            qDebug() << parameter;

            elem_size = std::min(bih.biWidth, bih.biHeight)/5;
            elem_size = std::max(elem_size, 80);


            elem->bih.biWidth = elem_size*2;
            elem->bih.biHeight = elem_size;
            elem->rgb = new RGB* [unsigned(elem->bih.biHeight)];
            for(int i = 0; i < elem->bih.biHeight; i++){
                elem->rgb[i] = new RGB [unsigned(elem->bih.biWidth)];
                for(int j = 0; j < elem->bih.biWidth; j++)
                    elem->rgb[i][j] = {255, 255, 255, 1};
            }
            double length = double(elem_size)/7.0;
            coordinate finish = {int(elem->bih.biWidth*0.6), int(elem-
>bih.biHeight*0.8)};
            coordinate start;
            coordinate tmp;
            double angle = parameter;

            for(int i = 1; length > 1; i++){
                length *= 0.8;
                start = finish;
                finish = {start.x +
static_cast<int>(round(length*sin(angle*i))), start.y -
static_cast<int>(round(length*cos(angle*i)))};
                if (i == 1) tmp = finish;
                elem->drawThickLine(start.x, start.y, finish.x, finish.y, width,
Qt::black);
            }

            length = double(elem_size)/5.0;
            finish = tmp;
            for(int i = 1; length > 3; i++){
                length *= 0.9;
                start = finish;
                finish = {start.x -
static_cast<int>(round(length*sin(angle*i/2))), start.y -
static_cast<int>(round(length*cos(angle*i/2)))};
```

```
                elem->drawThickLine(start.x, start.y, finish.x, finish.y, width,
Qt::black);
            }

            int base_move = -1;
            int side_move = elem->bih.biWidth;
            for(int i = elem->bih.biHeight - 1; i >= 0; i--){
                for(int j = 0; j < elem->bih.biWidth; j++){
                    if(elem->rgb[i][j].red == 0 && elem->rgb[i][j].green == 0 &&
elem->rgb[i][j].blue == 0){
                        if (i > base_move) base_move = i;
                        if (j < side_move) side_move = j;
                    }
                }
            }
            if (base_move == -1) base_move = 0;
            else base_move = elem->bih.biHeight - base_move - 1;

            if (side_move == elem->bih.biWidth) side_move = 0;


            int count;
            int branch_edge;

            branch_edge = elem->bih.biWidth/4;
            count = bih.biWidth/(2*branch_edge);

            //bottom left
            for(int i = 0; i < count; i ++){
                for(int y = 0; y < elem->bih.biHeight; y++)
                    for(int x = 0; x < elem->bih.biWidth; x++)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[y + (bih.biHeight - elem->bih.biHeight)
+ base_move][x + i*branch_edge - side_move].visible = 1;
                        }
            }

            //bottom right
            for(int i = 0; i < count; i++){
                for(int y = 0; y < elem->bih.biHeight; y++)
                    for(int x = elem->bih.biWidth - 1; x >= 0; x--)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[y + (bih.biHeight - elem->bih.biHeight)
+ base_move][bih.biWidth - 1 - x - i*branch_edge + side_move].visible = 1;
                        }
            }

            //top left
            for(int i = 0; i < count ; i++){
                for(int y = elem->bih.biHeight - 1; y >= 0; y--)
                    for(int x = 0; x < elem->bih.biWidth; x++)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[elem->bih.biHeight - 1 - y - base_move]
[x + i*branch_edge - side_move].visible = 1;
                        }
            }

            //top right
            for(int i = 0; i < count; i++){
                for(int y = elem->bih.biHeight - 1; y >= 0; y--)
                    for(int x = elem->bih.biWidth - 1; x >= 0; x--)
```

```
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[elem->bih.biHeight - 1 - y - base_move]
[bih.biWidth - 1 - x - i*branch_edge + side_move].visible = 1;
                        }
            }


            count = bih.biHeight/(2*branch_edge);

            //right top
            for(int i = 0; i < count; i++)
                for(int y = 0; y < elem->bih.biHeight; y++)
                    for(int x = 0; x < elem->bih.biWidth; x++)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[x - side_move + i*branch_edge][y +
bih.biWidth - elem->bih.biHeight + base_move].visible = 1;
                        }

            //right bottom
            for(int i = 0; i < count; i++){
                for(int y = 0; y < elem->bih.biHeight; y++)
                    for(int x = elem->bih.biWidth - 1; x >= 0; x--)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[bih.biHeight - 1 - x - i*branch_edge +
side_move][y + bih.biWidth - elem->bih.biHeight + base_move].visible = 1;
                        }
            }

            //left top
            for(int i = 0; i < count; i++){
                for(int y = elem->bih.biHeight - 1; y >= 0; y--)
                    for(int x = 0; x < elem->bih.biWidth; x++)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[x - side_move + i*branch_edge][elem-
>bih.biHeight - 1 - y - base_move].visible = 1;
                        }
            }

            //left bottom
            for(int i = 0; i < count; i++){
                for(int y = elem->bih.biHeight - 1; y >= 0; y--)
                    for(int x = elem->bih.biWidth - 1; x >= 0; x--)
                        if (elem->rgb[y][x].red == 0 && elem->rgb[y][x].green ==
0 && elem->rgb[y][x].blue == 0){
                            buffer->rgb[bih.biHeight - 1 - x - i*branch_edge +
side_move][elem->bih.biHeight - 1 - y - base_move].visible = 1;
                        }
            }
        }
    }

    else
        return 1;

    for(int i = 0; i < bih.biHeight; i++)
        for(int j = 0; j < bih.biWidth; j++)
            if(buffer->rgb[i][j].visible)
                rgb[i][j] = buffer->rgb[i][j];
    return 0;
}
```

newimagedialog.h

```
#ifndef NEWIMAGEDIALOG_H
#define NEWIMAGEDIALOG_H

#include <QDialog>

namespace Ui {
class NewImageDialog;
}

class NewImageDialog : public QDialog
{
    Q_OBJECT

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

public:
    int32_t width = 0;
    int32_t height = 0;

private slots:
    void on_Cancel_clicked();
    void on_Apply_clicked();

private:

    Ui::NewImageDialog *newimage;
};

#endif // NEWIMAGEDIALOG_H
```

newimagedialog.cpp

```
#include "newimagedialog.h"
#include "ui_newimagedialog.h"

NewImageDialog::NewImageDialog(QWidget *parent) :
    QDialog(parent),
    newimage(new Ui::NewImageDialog)
{
    newimage->setupUi(this);
    this->setWindowTitle("Новое изображение");
}

NewImageDialog::~NewImageDialog()
{
    delete newimage;
}


void NewImageDialog::on_Cancel_clicked()
{
    this->done(0);
}

void NewImageDialog::on_Apply_clicked()
{
    this->width = newimage->width->value();
    this->height = newimage->height->value();
    this->done(1);
}
```

createframedialog.h

```
#ifndef CREATEFRAMEDIALOG_H
#define CREATEFRAMEDIALOG_H

#include <QDialog>
#include "grapher.h"

#define OFF 0
#define BRANCH 1
#define RIM 2
#define CROSS 3
#define SQR_SIDE 88

namespace Ui {
class CreateFrameDialog;
}

class CreateFrameDialog : public QDialog
{
    Q_OBJECT

public:
    double radius = 0;
    int density = 1;
    double angle = 30 * PI / 180;
    int func = OFF;
    explicit CreateFrameDialog(QWidget *parent = nullptr);
    ~CreateFrameDialog();


private slots:
    void on_apply_clicked();
    void on_reject_clicked();

    void on_branches_clicked();
    void on_angle_value_valueChanged(int value);

    void on_line_frame_clicked();
    void on_density_value_valueChanged(int value);

    void on_kelt_cross_clicked();
    void on_radius_value_valueChanged(int value);

private:
    Grapher *map;
    Ui::CreateFrameDialog *ui;
    void drawBranch(double angle);
    void drawLineFrame(int density);
    void drawKeltCross(double radius);
};

#endif // CREATEFRAMEDIALOG_H
```

createframedialog.cpp

```
#include "createframedialog.h"
#include "ui_createframedialog.h"

#include <QDebug>


CreateFrameDialog::CreateFrameDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::CreateFrameDialog)
{
```

```
    ui->setupUi(this);
    ui->angle_value->setEnabled(0);
    ui->density_value->setEnabled(0);

    map = new Grapher;
    map->setFixedSize(170,170);
    map->scene->setBackgroundBrush(Qt::white);
    map->scene->setSceneRect(0, 0, 160, 160);
    this->ui->Field->addWidget(map);

    this->setWindowTitle("Создать рамку");
}

CreateFrameDialog::~CreateFrameDialog()
{
    delete ui;
}


void CreateFrameDialog::drawBranch(double angle){

    foreach(QGraphicsItem *item, map->scene->items())
        delete item;

    QPen pen(Qt::black, 4);
    pen.setJoinStyle(Qt::RoundJoin);
    pen.setCapStyle(Qt::RoundCap);
    double length = 25;
    coordinate start;
    coordinate finish = {110, 140};
    coordinate tmp;

    for(int i = 1; length > 3; i++){
        length *= 0.8;
        start = finish;
        finish = {start.x + static_cast<int>(round(length*sin(angle*i))),
start.y - static_cast<int>(round(length*cos(angle*i)))};
        if (i == 1) tmp = finish;
        map->scene->addLine(start.x, start.y, finish.x, finish.y, pen);
    }

    length = 25;
    finish = tmp;
    for(int i = 1; length > 5; i++){
        length *= 0.9;
        start = finish;
        finish = {start.x - static_cast<int>(round(length*sin(angle*i/2))),
start.y - static_cast<int>(round(length*cos(angle*i/2)))};
        map->scene->addLine(start.x, start.y, finish.x, finish.y, pen);
    }
}


void CreateFrameDialog::drawLineFrame(int density){
    foreach(QGraphicsItem *item, map->scene->items())
        delete item;

    QPen pen(Qt::black);
    pen.setJoinStyle(Qt::MiterJoin);
    int frame_count = 0;
    int width = 0;

    switch (density){
    case 1:
```

```
        width = 16;
        frame_count = 1;
        break;
    case 2:
        width = 12;
        frame_count = 2;
        break;
    case 3:
        width = 8;
        frame_count = 3;
        break;
    case 4:
        width = 6;
        frame_count = 4;
        break;
    case 5:
        width = 4;
        frame_count = 5;
        break;
    }
    pen.setWidth(width);

    int s_x = width/2;
    int s_y = width/2;
    int f_x = 168 - width;
    int f_y = 168 - width;

    for(int i = 0; i < frame_count; i++){
        map->scene->addRect(s_x, s_y, f_x, f_y, pen);
        s_x += width*3/2;
        s_y += width*3/2;
        f_x -= width*3;
        f_y -= width*3;
    }
}


void CreateFrameDialog::drawKeltCross(double radius){
    foreach(QGraphicsItem *item, map->scene->items())
        delete item;

    QPen pen(Qt::black, 5);
    pen.setCapStyle(Qt::FlatCap);

    map->scene->addLine(40, 40, 40+SQR_SIDE, 40+SQR_SIDE, pen);
    map->scene->addLine(40, 40+SQR_SIDE, 40+SQR_SIDE, 40, pen);
    map->scene->addEllipse(40+(SQR_SIDE/2-radius), 40+(SQR_SIDE/2-radius),
2*radius, 2*radius, pen);
}


void CreateFrameDialog::on_branches_clicked()
{
    func = BRANCH;
    ui->angle_value->setEnabled(1);
    ui->density_value->setEnabled(0);
    ui->radius_value->setEnabled(0);
    drawBranch(angle);
}

void CreateFrameDialog::on_angle_value_valueChanged(int value)
{
    angle = (PI * static_cast<double>(value))/180.0;
    drawBranch(angle);
```

```
}


void CreateFrameDialog::on_line_frame_clicked()
{
    func = RIM;
    ui->density_value->setEnabled(1);
    ui->angle_value->setEnabled(0);
    ui->radius_value->setEnabled(0);
    drawLineFrame(density);

}

void CreateFrameDialog::on_density_value_valueChanged(int value)
{
    density = value;
    drawLineFrame(density);
}


void CreateFrameDialog::on_kelt_cross_clicked()
{
    func = CROSS;
    ui->radius_value->setEnabled(1);
    ui->angle_value->setEnabled(0);
    ui->density_value->setEnabled(0);
    drawKeltCross(radius);
}

void CreateFrameDialog::on_radius_value_valueChanged(int value)
{
    radius = static_cast<double>(SQR_SIDE*value)/200.0;
    drawKeltCross(radius);
}


void CreateFrameDialog::on_apply_clicked()
{
    this->done(1);
}

void CreateFrameDialog::on_reject_clicked()
{
    this->done(0);
}
```

imageinfo.h

```
#ifndef IMAGEINFO_H
#define IMAGEINFO_H

#include <QDialog>
#include <QString>
#include "image.h"

namespace Ui {
class ImageInfo;
}

class ImageInfo : public QDialog
{
    Q_OBJECT

public:
    void update(Image *img);
```

```
    explicit ImageInfo(QWidget *parent = nullptr);
    ~ImageInfo();

private:
    Ui::ImageInfo *ui;
};

#endif // IMAGEINFO_H
```

imageinfo.cpp

```
#include "imageinfo.h"
#include "ui_imageinfo.h"

ImageInfo::ImageInfo(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::ImageInfo)
{
    ui->setupUi(this);
    ui->width->setBackgroundRole(QPalette::Base);
    ui->height->setBackgroundRole(QPalette::Base);
    ui->size->setBackgroundRole(QPalette::Base);
}

ImageInfo::~ImageInfo()
{
    delete ui;
}


void ImageInfo::update(Image *img){
    ui->width->setText("  " + QString::number(img->bih.biWidth) + " px");
    ui->height->setText("  " + QString::number(img->bih.biHeight) + " px");
    ui->size->setText("  " + QString::number(double(img->bfh.bfSize)/1024.0) + "
KB");
}
```

about.h

```
#ifndef ABOUT_H
#define ABOUT_H

#include <QDialog>

namespace Ui {
class About;
}

class About : public QDialog
{
    Q_OBJECT

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

private:
    Ui::About *ui;
};

#endif // ABOUT_H
```

about.cpp

```
#include "about.h"
#include "ui_about.h"

About::About(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::About)
{
    ui->setupUi(this);
}

About::~About()
{
    delete ui;
}
```