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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 3344 _____ Коршунов П.И.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Коршунов П.И.

Группа 3344

Тема работы: Обработка изображений.

Исходные данные:

- Программу требуется реализовать в виде утилиты, подобной стандартным *linux*-утилитам.
- Программа должна считать *bmp*-файл без сжатия с 24 битами на цвет
- Программа должна сохранить обработанный *bmp*-файл
- Все поля стандартных *BMP* заголовков в выходном файле должны иметь те же значения что и во входном
- Необходимо использовать *Makefile* для сборки проекта, название исполняемого файла должно быть: *cw*.

Содержание пояснительной записки:

- Содержание
- Введение
- Описание варианта работы
- Описание функций программы
- Описание структуры файлов программы
- Описание сборки проекта
- Примеры работы программы
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 45 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 15.05.2024

Студент _____ Коршунов П.И.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Программа для обработки изображений на языке *C++*, основанная на структурах *Image* и *Rgb*, реализованная в виде утилиты. Программа предоставляет набор инструментов для фильтрации, рисования и преобразования изображений. Фильтры включают в себя изменение значения *RGB*-компоненты, поиск и замену самого часто встречающегося цвета. Рисование включает в себя создание квадратов с заданными параметрами и заливку их цветом. Преобразование включает в себя обмен местами частей выбранной области. Операции выполняются с использованием динамической памяти и структур данных *Image* и *Rgb*. Результатом работы программы является обработанное изображение с учетом выполненных операций.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель проекта — изучение формата файла BMP и реализация утилиты на языке *C++* для работы с этим форматом. Задачи включают изучение структуры файла *BMP*, получение информацию об изображении, такую как его размеры и содержимое, обработку массива пикселей в соответствии с заданием, обработку исключительных случаев, таких как отсутствие файла или неверный формат, и сохранение итогового изображения в новый файл. Методы будут включать в себя реализацию функций для чтения и записи файлов BMP, а также функций для обработки изображений.

1. ОПИСАНИЕ ВАРИАНТА РАБОТЫ

Программа должна иметь следующую функции по обработке изображений:
Фильтр *rgb*-компонент. Флаг для выполнения данной операции: `*--rgbfilter*`. Этот инструмент должен позволять для всего изображения либо установить в диапазоне от 0 до 255 значение заданной компоненты. Функционал определяется
Какую компоненту требуется изменить. Флаг `*--component_name*`. Возможные значения `*red*`, `*green*` и `*blue*`.
В какой значение ее требуется изменить. Флаг `*--component_value*`. Принимает значение в виде числа от 0 до 255
Рисование квадрата. Флаг для выполнения данной операции: `--square`. Квадрат определяется:
Координатами левого верхнего угла. Флаг `*--left_up*`, значение задаётся в формате `*left.up*`, где *left* – координата по x, *up* – координата по y
Размером стороны. Флаг `*--side_size*`. На вход принимает число больше 0
Толщиной линий. Флаг `*--thickness*`. На вход принимает число больше 0
Цветом линий. Флаг `--color` (цвет задаётся строкой `*rrr.ggg.bbb*`, где *rrr/ggg/bbb* – числа, задающие цветовую компоненту. пример `*--color 255.0.0*` задаёт красный цвет)
Может быть залит или нет. Флаг `*--fill*`. Работает как бинарное значение: флага нет – *false* , флаг есть – *true*.
Цветом которым он залит, если пользователем выбран залитый. Флаг `*--fill_color*` (работает аналогично флагу `*--color*`)
Поменять местами 4 куска области. Флаг для выполнения данной операции: `*--exchange*`. Выбранная пользователем прямоугольная область делится на 4 части и эти части меняются местами. Функционал определяется:
Координатами левого верхнего угла области. Флаг `*--left_up*`, значение задаётся в формате `*left.up*`, где *left* – координата по x, *up* – координата по y

Координатами правого нижнего угла области. Флаг `--*right_down*`, значение задаётся в формате `*right.down*`, где *right* – координата по x, *down* – координата по y

Способом обмена частей: “по кругу”, по диагонали. Флаг `--*exchange_type*`, возможные значения: `*clockwise*`, `*counterclockwise*`, `*diagonals*`

Находит самый часто встречаемый цвет и заменяет его на другой заданный цвет. Флаг для выполнения данной операции: `--*freq_color*`. Функционал определяется:

Цветом, в который надо перекрасить самый часто встречаемый цвет. Флаг `--*color*` (цвет задаётся строкой `*rrr.ggg.bbb*`, где *rrr/ggg/bbb* – числа, задающие цветовую компоненту. пример `--*color* 255.0.0` задаёт красный цвет)

Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Сборка должна осуществляться при помощи *make* и *Makefile* или другой системы сборки

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Описание функций программы

В программе используется класс *Image*, которая представляет изображение и включает в себя заголовок файла *BitmapFileHeader*, заголовок информации *BitmapInfoHeader*, высоту *H* и ширину *W* изображения, а также указатель на массив пикселей *arr*. Также используется структура *Rgb*, которая представляет из себя 3 числа *r, g, b*, отвечающих за 3 канала пикселя.

Функции и их краткое описание:

- Image(string bmp_name)
  Конструктор класса *Image*, создает объект изображения из файла *bmp*.
- ~Image()
  Деструктор класса *Image*, освобождает память, выделенную под массив пикселей.
- void printFileHeader()
  Выводит на экран информацию о заголовке файла BMP.
- void printInfoHeader()
  Выводит на экран информацию о заголовке информации BMP.
- void set_pixel(Rgb &pix, Rgb new_pix)
  Устанавливает новый цвет пикселю.
- void swap_pixel(Rgb &pix, Rgb &new_pix)
  Меняет цвета двух пикселей местами.
- void fill_zone(int i, int j, Rgb border)
  Заливает область изображения, ограниченную границей border.
- void plort_circle(int x, int y, int x0, int y0, Rgb new_pix, int thickness)
  Вспомогательная функция для рисования окружности.

- void draw_circle(int y0, int x0, int radius, Rgb new_pix, bool isfill = false, int thickness = 0, bool fillcolor = false, Rgb new_in_pix = {0, 0, 0})

  Рисует окружность с центром в точке (*x0, y0*) радиусом *radius* и заливает ее цветом, если *isfill == true*.
- void draw_line(int x1, int y1, int x2, int y2, Rgb &new_pix, int thickness = 0)

  Рисует линию между точками (*x1, y1*) и (*x2, y2*) цветом *new_pix* и толщиной *thickness*.
- void rgbfilter(string comp_n, int comp_v)

  Применяет фильтр к изображению, изменяя значение одного из компонентов *RGB*.
- void draw_square(int h, int w, int side_size, Rgb new_pix, int thickness = 1, bool isfill = false, Rgb new_in_pix = {0, 0, 0})

  Рисует квадрат с левым верхним углом в точке *w, h*, стороной *side_size* и толщиной *thickness* и заливает его цветом, если *isfill == true*.
- void exchange(int left_up[], int right_down[], string exchange_type)

  Обменивает части изображения, заданные прямоугольником с левым верхним углом *left_up* и правым нижним углом *right_down*, в зависимости от типа обмена *exchange_type*.
- Rgb get_frequent()

  Возвращает наиболее часто встречающийся цвет на изображении.
- void swap_frequent(Rgb new_pix)

  Меняет наиболее часто встречающийся цвет на изображении на новый цвет *new_pix*.
- Rgb **read_bmp(string bmp_name)

  Читает изображение из файла *bmp* и возвращает массив пикселей.
- void write_bmp(string bmp_name)

Записывает изображение в файл *bmp*.

- bool check_coords(int y, int x)
  Проверяет, выходит ли точка за пределы изображения.
- void print_usage()
  Выводит на экран информацию о возможных опциях командной строки.
- bool is_integer(const std::string &str)
  Проверяет, является ли строка *str* целым числом.
- Rgb create_Rgb(int b_in, int g_in, int r_in)
  Создает объект *Rgb* с заданными значениями компонентов.
- int main(int argc, char *argv[])
  Основная функция программы, которая считывает опции командной строки, создает объект изображения, выполняет необходимые операции и записывает изображение в файл.

2.2. Описание структуры файлов программы

В программе предусмотрены следующие файлы:

- *Makefile*: Файл для автоматизации процесса компиляции и сборки программы.

- *image_bmp.h*: Заголовочный файл, содержащий определение класса Image, который представляет изображение в формате BMP и предоставляет методы для работы с ним.

- *image_io.cpp*: Файл с реализацией методов класса *Image*, связанных с чтением и записью изображений в формате *BMP*.

- *image_processing.cpp*: Файл с реализацией методов класса *Image*, связанных с обработкой изображений, таких как фильтрация, рисование фигур и т.д.

- *input.cpp*: Файл с реализацией функций для чтения данных из консоли и файлов.

- *input.h:* Заголовочный файл, содержащий прототипы функций для чтения данных.

- *structs.h:* Заголовочный файл, содержащий определения структур данных, таких как *Rgb* для представления цвета пикселя и *BitmapInfoHeader, BitmapFileHeader* для представления заголовка файла *BMP*.

- *main.cpp*: Файл, содержащий функцию *main*, отвечающую за взаимодействие с пользователем и связку всех остальных функций.

Эта структура файлов позволяет организовать код программы в логически связанные блоки, что облегчает его понимание и поддержку.

2.3. Описание сборки проекта

Для сборки проекта использовался *Makefile*, содержащий инструкции компиляции и компоновки. Проект разделен на несколько модулей:

- *main.o:* Главный модуль, отвечающий за управление программой.
- *image_io.o*: Модуль, реализующий чтение и запись изображений в формате *BMP*.

- *image_processing.o*: Модуль, содержащий функции для обработки изображений.
- *input.o*: Модуль для ввода данных.

Компилятор: *g++*

Флаги:

- *-std=c++11*

  Устанавливает стандарт языка C++ на уровне C++11.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Пример 1: Рисование квадрата.

Ввод	Вывод
-s --fill_color 38.102.253 --thickness 8 --left_up 308.426 --output ./output.bmp --input ./input.bmp --color 209.189.245 --fill --side_size 213	Правильно измененная картинка

Пример 2: Изменение значения одной компоненты.

Ввод	Вывод
-r --component_name green --component_value 100 input.bmp	Правильно измененная картинка

Пример 3: Изменение частей прямоугольной области.

Ввод	Вывод
-e --left_up 30.40 --right_down 100.200 --exchange_type clockwise input.bmp	Правильно измененная картинка

Пример 4: Изменение самого частого цвета.

Ввод	Вывод
-f --color 240.1.1 input.bmp	Правильно измененная картинка

Пример 5: Вывод информации об изображении.

Ввод	Вывод
-I input.bmp	Course work for option 5.3, created by Petr Korshunov signature: 0x4d42 (19778) filesize: 0x141a62 (1317474)

	reserved1: 0x0 (0) reserved2: 0x0 (0) pixelArrOffset: 0x36 (54) headerSize: 0x28 (40) width: 0x30c (780) height: 0x233 (563) planes: 0x1 (1) bitsPerPixel: 0x18 (24) compression: 0x0 (0) imageSize: 0x0 (0) xPixelsPerMeter: 0x2e23 (11811) yPixelsPerMeter: 0x2e23 (11811) colorsInColorTable: 0x0 (0) importantColorCount: 0x0 (0)

Пример 6: Проверка обработки ошибок.

Ввод	Вывод
-r --component_name green --component_value 100.100.100 input.bmp	Course work for option 5.3, created by Petr Korshunov Error in parsing arguments: --component_value argument is incorrect

ЗАКЛЮЧЕНИЕ

Была успешно реализована программа на языке C++ для обработки изображений в формате BMP. Программа выполняет поставленные задачи, включая чтение и запись изображений, фильтрацию, преобразование цвета, и др. Полученные результаты подтверждают успешное достижение поставленной цели. В ходе выполнения работы были приобретены навыки работы с изображениями, создания Makefile для сборки проекта, использования структур данных, работы с функциями стандартной библиотеки C++, а также оформления кода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Cplusplus. URL: https://cplusplus.com/reference/ (Дата обращения 12.05.2024)
2. Базовые сведения к выполнению курсовой и лабораторных работ по дисциплине «программирование». Второй семестр: учеб.-метод. пособие др. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2024. 36 с.
3. Geeksforgeeks. URL: https://www.geeksforgeeks.org (Дата обращения 10.05.2024)

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

main.cpp

```
#include "structs.h"
#include "image_bmp.h"
#include "input.h"

int main(int argc, char *argv[])
{
    cout << "Course work for option 5.3, created by Petr Korshunov"
<< endl;

    Config config = get_config(argc, argv);

    if (!config.name_in_in)
    {
        config.name_in_in = 1;
        config.name_in = argv[argc - 1];
    }

    if (argc == 1)
    {
        print_usage();
        return 0;
    }

    if (config.name_in == config.name_out)
    {
        cout << "Error in main: input and output file names cannot
be the same" << endl;
        exit(41);
    }

    if (config.name_in_in)
    {
        Image img(config.name_in);

        if (config.info == 1)
        {
            img.printFileHeader();
```

```
                img.printInfoHeader();
                return 0;
            }
            else if (config.rgb_filter && config.rgb_name_in &&
config.rgb_val_in)
            {
                img.rgbfilter(config.rgb_name, config.rgb_val);
            }
            else if (config.square && config.sq_color_in &&
config.sq_side_in && config.sq_thickness_in && config.sq_lu_in)
            {
                img.draw_square(config.sq_lu[0], config.sq_lu[1],
config.sq_side, config.sq_color, config.sq_thickness, config.sq_fill,
config.sq_fill_color);
            }
            else if (config.exchange && config.ex_lu_in &&
config.ex_rd_in && config.ex_type_in)
            {
                img.exchange(config.ex_lu, config.ex_rd,
config.ex_type);
            }
            else if (config.frequent && config.freq_color_in)
            {
                img.swap_frequent(config.freq_color);
            }

            img.write_bmp(config.name_out);
        }
        else
        {
            cout << "Error in parsing arguments: --input argument does
not exist" << endl;
            exit(41);
        }
        return 0;
    }
```

input.cpp

```
#include "input.h"
#include "structs.h"

void print_usage()
{
    cout << "Options:\n";
```

```
        cout << "  -i, --input <file>         Input file name\n";
        cout << "  -o, --output <file>        Output file name\n";
        cout << "  -r, --rgbfilter            Enable RGB filter\n";
        cout << "   --component_name <name>     Name of RGB component to
change\n";
        cout << "  --component_value <value>  Value of RGB component to
change\n";
        cout << "  -s, --square               Enable square drawing\n";
        cout << "   --left_up <x,y>             Square left upper corner
coordinates\n";
        cout << "  --side_size <size>         Square side size\n";
        cout << "  --thickness <thickness>    Square side thickness\n";
        cout << "  --color <r,g,b>            Square color\n";
        cout << "  --fill                   Enable/disable        square
filling\n";
        cout << "  --fill_color <r,g,b>       Square fill color\n";
        cout << "  -e, --exchange           Enable exchange operation\n";
        cout << "  --left_up <x,y>            Exchange left upper corner
coordinates\n";
        cout << "  --right_down <x,y>         Exchange right down corner
coordinates\n";
        cout  <<  "    --exchange_type  <type>          Exchange  type
(clockwise|counterclockwise|diagonals)\n";
        cout << "   -f, --freq_color             Enable frequent color
swaping\n";
        cout << "  --color <r,g,b>            Color to swap\n";
        cout << "  -h, --help                 Show this help message\n";
        cout << "  -I, --info                 Info about this image\n";
    }

    bool is_integer(const string &str)
    {
        if (str.empty())
        {
            return false;
        }
        for (char c : str)
        {
            if (!std::isdigit(c))
            {
                return false;
            }
        }
        return true;
```

```
    }

    Rgb create_Rgb(int b_in, int g_in, int r_in)
    {
        if (0 <= b_in <= 255 && 0 <= g_in <= 255 && 0 <= r_in <= 255)
        {
            unsigned char b = (unsigned char)b_in;
            unsigned char g = (unsigned char)g_in;
            unsigned char r = (unsigned char)r_in;
            return Rgb{b, g, r};
        }
        else
        {
            cout << "Error in create_Rgb: some of the arguments are
incorrect" << endl;
            exit(45);
        }
    }

    Config get_config(int argc, char *argv[])
    {
        Config config = {"out.bmp"};
        string opts = "i:o:Ihrsef";
        option longOpts[] = {
            {"help", no_argument, NULL, 'h'},
            {"info", no_argument, NULL, 'I'},
            {"input", required_argument, NULL, 'i'},
            {"output", required_argument, NULL, 'o'},
            {"rgbfilter", no_argument, NULL, 'r'},
            {"component_name", required_argument, NULL, 0},
            {"component_value", required_argument, NULL, 0},
            {"square", no_argument, NULL, 's'},
            {"left_up", required_argument, NULL, 0},
            {"side_size", required_argument, NULL, 0},
            {"thickness", required_argument, NULL, 0},
            {"color", required_argument, NULL, 0},
            {"fill", no_argument, NULL, 0},
            {"fill_color", required_argument, NULL, 0},
            {"exchange", no_argument, NULL, 'e'},
            {"left_up", required_argument, NULL, 0},
            {"right_down", required_argument, NULL, 0},
            {"exchange_type", required_argument, NULL, 0},
            {"freq_color", no_argument, NULL, 'f'},
            {"color", required_argument, NULL, 0},
```

```
            {NULL, 0, NULL, 0}};

        int opt;
        int longIndex;
        opt   =   getopt_long(argc,   argv,   opts.c_str(),   longOpts,
&longIndex);

        while (opt != -1)
        {
            switch (opt)
            {
            case 'i':
                config.name_in_in = 1;
                config.name_in = optarg;
                break;
            case 'o':
                config.name_out_in = 1;
                config.name_out = optarg;
                break;
            case 'r':
                config.rgb_filter = 1;
                break;
            case 's':
                config.square = 1;
                break;
            case 'e':
                config.exchange = 1;
                break;
            case 'f':
                config.frequent = 1;
                break;
            case 'I':
                config.info = 1;
                break;
            case 'h':
                print_usage();
                exit(0);
            case 0:
                if (longOpts[longIndex].name == "component_name")
                    {
                          config.rgb_name_in = 1;
                          config.rgb_name = optarg;
                    }
```

```
                    else      if      (longOpts[longIndex].name      ==
"component_value")
                    {
                          config.rgb_val_in = 1;
                          if (is_integer(optarg))
                          {
                                config.rgb_val = stoi(optarg);
                          }
                          else
                          {
                                cout << "Error in parsing arguments: --
component_value argument is incorrect" << endl;
                                exit(41);
                          }
                    }
                    else if (longOpts[longIndex].name == "left_up")
                    {
                          config.sq_lu_in = 1;
                          config.ex_lu_in = 1;
                          if        (is_integer(string(optarg).substr(0,
string(optarg).find('.')))                                          &&
is_integer(string(optarg).substr(string(optarg).find('.') + 1)))
                          {
                                int  x  =  stoi(string(optarg).substr(0,
string(optarg).find('.')));
                                int                  y                  =
stoi(string(optarg).substr(string(optarg).find('.') + 1));
                                config.sq_lu[0] = y;
                                config.sq_lu[1] = x;
                                config.ex_lu[0] = y;
                                config.ex_lu[1] = x;
                          }
                          else
                          {
                                cout << "Error in parsing arguments: --
left_up argument is incorrect" << endl;
                                exit(41);
                          }
                    }
                    else if (longOpts[longIndex].name == "side_size")
                    {
                          config.sq_side_in = 1;
                          if (is_integer(optarg))
                          {
```

```
                              config.sq_side = stoi(optarg);
                        }
                        else
                        {
                              cout << "Error in parsing arguments: --
side_size argument is incorrect" << endl;
                              exit(41);
                        }
                  }
                  else if (longOpts[longIndex].name == "thickness")
                  {
                        config.sq_thickness_in = 1;
                        if (is_integer(optarg))
                        {
                              config.sq_thickness = stoi(optarg);
                        }
                        else
                        {
                              cout << "Error in parsing arguments: --
thickness argument is incorrect" << endl;
                              exit(41);
                        }
                  }
                  else if (longOpts[longIndex].name == "color")
                  {
                        config.sq_color_in = 1;
                        config.freq_color_in = 1;
                  size_t first_dot = string(optarg).find('.');
                        size_t  second_dot  =  string(optarg).find('.',
first_dot + 1);
                  string r_str = string(optarg).substr(0, first_dot);
                  string g_str = string(optarg).substr(first_dot + 1,
second_dot - first_dot - 1);
                  string  b_str  =  string(optarg).substr(second_dot  +
1);
                        if (is_integer(r_str) && is_integer(g_str) &&
is_integer(b_str))
                        {
                              int r = stoi(r_str);
                              int g = stoi(g_str);
                              int b = stoi(b_str);
                              config.sq_color = create_Rgb(b, g, r);
                              config.freq_color = create_Rgb(b, g, r);
                        }
```

```
                        else
                        {
                              cout << "Error in parsing arguments: --
color argument is incorrect" << endl;
                              exit(41);
                        }
                  }
                  else if (longOpts[longIndex].name == "fill")
                  {
                        config.sq_fill = 1;
                  }
                  else if (longOpts[longIndex].name == "fill_color")
                  {
                        config.sq_fill_color_in = 1;
                  size_t first_dot = string(optarg).find('.');
                        size_t  second_dot  =  string(optarg).find('.',
first_dot + 1);
                  string r_str = string(optarg).substr(0, first_dot);
                  string g_str = string(optarg).substr(first_dot + 1,
second_dot - first_dot - 1);
                  string  b_str  =  string(optarg).substr(second_dot  +
1);
                        if (is_integer(r_str) && is_integer(g_str) &&
is_integer(b_str))
                        {
                              int r = stoi(r_str);
                              int g = stoi(g_str);
                              int b = stoi(b_str);
                              config.sq_fill_color  =  create_Rgb(b,  g,
r);
                        }
                        else
                        {
                              cout << "Error in parsing arguments: --
fill_color argument is incorrect" << endl;
                              exit(41);
                        }
                  }
                  else if (longOpts[longIndex].name == "right_down")
                  {
                        config.ex_rd_in = 1;
                        if        (is_integer(string(optarg).substr(0,
string(optarg).find('.')))                                          &&
is_integer(string(optarg).substr(string(optarg).find('.') + 1)))
```

```
                        {
                            int  x  =  stoi(string(optarg).substr(0,
string(optarg).find('.')));
                            int                 y                 =
stoi(string(optarg).substr(string(optarg).find('.') + 1));
                            config.ex_rd[0] = y;
                            config.ex_rd[1] = x;
                        }
                        else
                        {
                            cout << "Error in parsing arguments: --
right_down argument is incorrect" << endl;
                            exit(41);
                        }
                    }
                    else      if      (longOpts[longIndex].name      ==
"exchange_type")
                    {
                        config.ex_type_in = 1;
                        config.ex_type = optarg;
                    }
                    else
                    {
                    cout << "Error in parsing arguments: invalid argument
'" << optarg << "'" << endl;
                    exit(41);
                }
            }
            opt  =  getopt_long(argc,  argv,  opts.c_str(),  longOpts,
&longIndex);
        }

        return config;
    }
```

input.h

```
#ifndef INPUT_H
#define INPUT_H

#include "structs.h"

void print_usage();

bool is_integer(const string &str);
```

```
Rgb create_Rgb(int b_in, int g_in, int r_in);

Config get_config(int argc, char *argv[]);

#endif
```

image_io.cpp

```
#include "structs.h"
#include "image_bmp.h"

void Image::printFileHeader()
{
    cout << "signature:    \t0x" << hex << bmfh.signature << " (" <<
dec << bmfh.signature << ")" << endl;
    cout << "filesize:     \t0x" << hex << bmfh.filesize << " (" <<
dec << bmfh.filesize << ")" << endl;
    cout << "reserved1:    \t0x" << hex << bmfh.reserved1 << " (" <<
dec << bmfh.reserved1 << ")" << endl;
    cout << "reserved2:    \t0x" << hex << bmfh.reserved2 << " (" <<
dec << bmfh.reserved2 << ")" << endl;
    cout << "pixelArrOffset:\t0x" << hex << bmfh.pixelArrOffset << "
(" << dec << bmfh.pixelArrOffset << ")" << endl;
}

void Image::printInfoHeader()
{
    cout << "headerSize:   \t0x" << hex << bmif.headerSize << " ("
<< dec << bmif.headerSize << ")" << endl;
    cout << "width:        \t0x" << hex << bmif.width << " (" << dec
<< bmif.width << ")" << endl;
    cout << "height:       \t0x" << hex << bmif.height << " (" << dec
<< bmif.height << ")" << endl;
    cout << "planes:       \t0x" << hex << bmif.planes << " (" << dec
<< bmif.planes << ")" << endl;
    cout << "bitsPerPixel: \t0x" << hex << bmif.bitsPerPixel << " ("
<< dec << bmif.bitsPerPixel << ")" << endl;
    cout << "compression:  \t0x" << hex << bmif.compression << " ("
<< dec << bmif.compression << ")" << endl;
    cout << "imageSize:    \t0x" << hex << bmif.imageSize << " (" <<
dec << bmif.imageSize << ")" << endl;
    cout << "xPixelsPerMeter:\t0x" << hex << bmif.xPixelsPerMeter <<
" (" << dec << bmif.xPixelsPerMeter << ")" << endl;
```

```
        cout << "yPixelsPerMeter:\t0x" << hex << bmif.yPixelsPerMeter <<
" (" << dec << bmif.yPixelsPerMeter << ")" << endl;
        cout     <<     "colorsInColorTable:\t0x"     <<     hex     <<
bmif.colorsInColorTable << " (" << dec << bmif.colorsInColorTable << ")"
<< endl;
        cout     <<     "importantColorCount:\t0x"     <<     hex     <<
bmif.importantColorCount << " (" << dec << bmif.importantColorCount << ")"
<< endl;
    }

    Rgb **Image::read_bmp(string bmp_name)
    {
        FILE *f = fopen(bmp_name.c_str(), "rb");

        if (!f)
        {
            cout  <<  "Error  in  read_bmp:  failed  to  open  "  <<
bmp_name.c_str() << endl;
            exit(46);
        }

        fread(&bmfh, 1, sizeof(BitmapFileHeader), f);
        fread(&bmif, 1, sizeof(BitmapInfoHeader), f);

        if  (bmfh.signature  !=  0x4d42  ||  bmif.compression  !=  0  ||
bmif.bitsPerPixel != 24)
        {
            cout  <<  "Error  in  read_bmp:  Invalid  BMP  file  format:
incorrect signature or unsupported compression or incorrect color depth"
<< endl;
            exit(46);
        }

        unsigned int H = bmif.height;
        unsigned int W = bmif.width;
        unsigned int padding = (W * sizeof(Rgb)) % 4;
        if (padding)
            padding = 4 - padding;

        Rgb **arr = new Rgb *[H];
        for (int i = 0; i < H; i++)
        {
            arr[H - i - 1] = new Rgb[W + padding];
            fread(arr[H - i - 1], 1, W * sizeof(Rgb) + padding, f);
```

```
        }

        fclose(f);

        return arr;
    }

    void Image::write_bmp(string bmp_name)
    {
        FILE *ff = fopen(bmp_name.c_str(), "wb");

        if (!ff)
        {
            cout << "Error in write_bmp: failed to open " <<
bmp_name.c_str() << endl;
            exit(47);
        }

        unsigned int H = bmif.height;
        unsigned int W = bmif.width;
        unsigned int padding = (W * sizeof(Rgb)) % 4;
        if (padding)
            padding = 4 - padding;

        fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
        fwrite(&bmif, 1, sizeof(BitmapInfoHeader), ff);
        unsigned int w = W * sizeof(Rgb) + padding;
        for (int i = 0; i < H; i++)
        {
            fwrite(arr[H - i - 1], 1, w, ff);
        }
        fclose(ff);
    }
```

image_processing.cpp

```
    #include "structs.h"
    #include "image_bmp.h"

    void Image::set_pixel(Rgb &pix, Rgb new_pix)
    {
        pix.r = new_pix.r;
        pix.g = new_pix.g;
        pix.b = new_pix.b;
    }
```

```
void Image::swap_pixel(Rgb &pix, Rgb &new_pix)
{
    Rgb t = pix;

    pix.r = new_pix.r;
    pix.g = new_pix.g;
    pix.b = new_pix.b;

    new_pix.r = t.r;
    new_pix.g = t.g;
    new_pix.b = t.b;
}

void Image::fill_zone(int i, int j, Rgb border)
{
    if (!check_coords(i, j) || arr[i][j] == border)
    {
        return;
    }
    set_pixel(arr[i][j], border);
    fill_zone(i + 1, j, border);
    fill_zone(i - 1, j, border);
    fill_zone(i, j + 1, border);
    fill_zone(i, j - 1, border);
}

void Image::plort_circle(int x, int y, int x0, int y0, Rgb new_pix,
int thickness)
{
    for (int i = -thickness; i <= thickness; i++)
    {
        for (int j = -thickness; j <= thickness; j++)
        {
            if (check_coords(y0 + y + i, x0 + x + j) && abs(i) <=
thickness && abs(j) <= thickness)
            {
                set_pixel(arr[y0 + y + i][x0 + x + j], new_pix);
            }
            if (check_coords(y0 + y + i, x0 - x - j) && abs(i) <=
thickness && abs(j) <= thickness)
            {
                set_pixel(arr[y0 + y + i][x0 - x - j], new_pix);
            }
```

```
                if (check_coords(y0 - y - i, x0 + x + j) && abs(i) <=
thickness && abs(j) <= thickness)
                {
                    set_pixel(arr[y0 - y - i][x0 + x + j], new_pix);
                }
                if (check_coords(y0 - y - i, x0 - x - j) && abs(i) <=
thickness && abs(j) <= thickness)
                {
                    set_pixel(arr[y0 - y - i][x0 - x - j], new_pix);
                }
            }
        }
    }

    // Модифицированный пример из книги Г.Шилдта «Си для профессиональных
программистов» (алгоритм мичнера)
    void Image::draw_circle(int y0, int x0, int radius, Rgb new_pix, bool
isfill /*=false*/, int thickness /*=0*/, bool fillcolor /*=false*/, Rgb
new_in_pix /*={0, 0, 0}*/)
    {
        if (!fillcolor)
        {
            new_in_pix = new_pix;
        }
        int x = 0;
        int y = radius;
        int delta = 3 - 2 * radius;

        if (isfill)
        {
            for (int i = y0 - radius; i <= y0 + radius; i++)
            {
                for (int j = x0 - radius; j <= x0 + radius; j++)
                {
                    if (check_coords(i, j) && sqrt(pow(i - y0, 2) + pow(j
- x0, 2)) <= radius)
                    {
                        set_pixel(arr[i][j], new_in_pix);
                    }
                }
            }
        }

        while (x < y)
```

```
        {
            plort_circle(x, y, x0, y0, new_pix, thickness);
            plort_circle(y, x, x0, y0, new_pix, thickness);

            if (delta < 0)
            {
                delta += 4 * x + 6;
            }
            else
            {
                delta += 4 * (x - y) + 10;
                y--;
            }
            x++;
        }
        if (x == y)
        {
            plort_circle(x, y, x0, y0, new_pix, thickness);
        }
    }

    void Image::draw_line(int x1, int y1, int x2, int y2, Rgb &new_pix,
int thickness /*=0*/)
    {
        const int deltaX = abs(x2 - x1);
        const int deltaY = abs(y2 - y1);
        const int signX = x1 < x2 ? 1 : -1;
        const int signY = y1 < y2 ? 1 : -1;
        int error = deltaX - deltaY;
        Rgb &pix = arr[y2][x2];
        if (thickness > 0)
        {
            for (int i = thickness / 2; i > 0; i--)
            {
                draw_circle(y2, x2, i, new_pix, true, 0);
            }
        }
        if (check_coords(y2, x2))
        {
            set_pixel(pix, new_pix);
        }
        while (x1 != x2 || y1 != y2)
        {
            Rgb &pix = arr[y1][x1];
```

```
            if (thickness > 0)
            {
                for (int i = thickness / 2; i > 0; i--)
                {
                    draw_circle(y1, x1, i, new_pix, true, 0);
                }
            }
            if (check_coords(y1, x1))
            {
                set_pixel(pix, new_pix);
            }
            int error2 = error * 2;
            if (error2 > -deltaY)
            {
                error -= deltaY;
                x1 += signX;
            }
            if (error2 < deltaX)
            {
                error += deltaX;
                y1 += signY;
            }
        }
    }

    void Image::rgbfilter(string comp_n, int comp_v)
    {
        if (comp_v < 0 || comp_v > 255)
        {
            cout << "Error in rgbfilter: comp_v is incorrect" << endl;
            exit(42);
        }

        if (comp_n != "red" && comp_n != "green" && comp_n != "blue")
        {
            cout << "Error in rgbfilter: comp_n is incorrect" << endl;
            exit(42);
        }

        for (int i = 0; i < H; i++)
        {
            for (int j = 0; j < W; j++)
            {
                Rgb &pix = arr[i][j];
```

```
                if (comp_n == "red" and check_coords(i, j))
                {
                    set_pixel(pix,    Rgb{pix.b,    pix.g,    (unsigned
char)comp_v});
                }
                else if (comp_n == "green" and check_coords(i, j))
                {
                    set_pixel(pix,  Rgb{pix.b,  (unsigned  char)comp_v,
pix.r});
                }
                else if (comp_n == "blue" and check_coords(i, j))
                {
                    set_pixel(pix,  Rgb{(unsigned  char)comp_v,  pix.g,
pix.r});
                }
            }
        }
    }

    void Image::draw_square(int h, int w, int side_size, Rgb new_pix, int
thickness /*=1*/, bool isfill /*=false*/, Rgb new_in_pix /*={0, 0, 0}*/)
    {
        if (side_size <= 0)
        {
            cout << "Error in draw_square: side_size is incorrect" <<
endl;
            exit(43);
        }
        if (thickness <= 0)
        {
            cout << "Error in draw_square: thickness is incorrect" <<
endl;
            exit(43);
        }

        side_size--;

        if (isfill)
        {
            for (int i = 1; i < side_size; i++)
            {
                draw_line(w, h + i, w + side_size, h + i, new_in_pix);
                // for(int j = 1; j<side_size; j++){
                //  set_pixel(arr[h+i][w+j], new_in_pix);
```

```
                //  set_pixel(arr[h+side_size-i][w+side_size-j],
new_in_pix);
                // }
            }
        }

        draw_line(w, h, w + side_size, h, new_pix, thickness - 1);
        draw_line(w, h, w, h + side_size, new_pix, thickness - 1);
        draw_line(w + side_size, h, w + side_size, h + side_size, new_pix,
thickness - 1);
        draw_line(w, h + side_size, w + side_size, h + side_size, new_pix,
thickness - 1);

        // for(int i=0, j = 0; i<=side_size; i++, j++){
        //          set_pixel(arr[h+i][w], new_pix);
        //          set_pixel(arr[h][w+j], new_pix);
        //          set_pixel(arr[h+side_size-i][w+side_size],
new_pix);
        //          set_pixel(arr[h+side_size][w+side_size-j],
new_pix);
        // }
    }

    void Image::exchange(int left_up[], int right_down[], string
exchange_type)
    {

        if (exchange_type != "clockwise" && exchange_type !=
"counterclockwise" && exchange_type != "diagonals")
        {
            cout << "Error in exchange: exchange_type is incorrect" <<
endl;
            exit(44);
        }

        int lu_y = left_up[0] < right_down[0] ? left_up[0] :
right_down[0];
        int lu_x = left_up[1] < right_down[1] ? left_up[1] :
right_down[1];
        int rd_y = left_up[0] >= right_down[0] ? left_up[0] :
right_down[0];
        int rd_x = left_up[1] >= right_down[1] ? left_up[1] :
right_down[1];
```

```
        int x_side = (rd_x - lu_x + (1 - (rd_x - lu_x) % 2)) / 2;
        int y_side = (rd_y - lu_y + (1 - (rd_y - lu_y) % 2)) / 2;

        // lu_y += (rd_y - lu_y + 1) % 2;
        // lu_x += (rd_x - lu_x + 1) % 2;

        Rgb **first_part = new Rgb *[y_side];
        Rgb **second_part = new Rgb *[y_side];
        Rgb **third_part = new Rgb *[y_side];
        Rgb **forth_part = new Rgb *[y_side];

        for (int i = 0; i < y_side; i++)
        {

            first_part[i] = new Rgb[x_side];
            second_part[i] = new Rgb[x_side];
            third_part[i] = new Rgb[x_side];
            forth_part[i] = new Rgb[x_side];

            for (int j = 0; j < x_side; j++)
            {
                if (check_coords(lu_y + i, lu_x + j))
                {
                    first_part[i][j] = arr[lu_y + i][lu_x + j];
                }
                if (check_coords(lu_y + i, x_side + lu_x + j))
                {
                    second_part[i][j] = arr[lu_y + i][x_side + lu_x + j];
                }
                if (check_coords(y_side + lu_y + i, x_side + lu_x + j))
                {
                    third_part[i][j] = arr[y_side + lu_y + i][x_side +
lu_x + j];
                }
                if (check_coords(y_side + lu_y + i, lu_x + j))
                {
                    forth_part[i][j] = arr[y_side + lu_y + i][lu_x + j];
                }

                // if (exchange_type == "diagonals")
                // {
                //  swap_pixel(arr[lu_y + i][lu_x + j], arr[y_side + lu_y
+ i][x_side + lu_x + j]);
```

```
                //  swap_pixel(arr[lu_y  +  i][x_side  +  lu_x  +  j],
arr[y_side + lu_y + i][lu_x + j]);
                // }
                // if (exchange_type == "clockwise")
                // {
                //  swap_pixel(arr[lu_y + i][lu_x + j], arr[y_side + lu_y
+ i][lu_x + j]);
                //  swap_pixel(arr[y_side  +  lu_y  +  i][lu_x  +  j],
arr[y_side + lu_y + i][x_side + lu_x + j]);
                //  swap_pixel(arr[y_side + lu_y + i][x_side + lu_x +
j], arr[lu_y + i][x_side + lu_x + j]);
                // }
                // if (exchange_type == "counterclockwise") {
                //  swap_pixel(arr[lu_y  +  i][lu_x  +  j],  arr[lu_y  +
i][x_side + lu_x + j]);
                //  swap_pixel(arr[lu_y  +  i][x_side  +  lu_x  +  j],
arr[y_side + lu_y + i][x_side + lu_x + j]);
                //  swap_pixel(arr[y_side + lu_y + i][x_side + lu_x +
j], arr[y_side + lu_y + i][lu_x + j]);
                // }
            }
        }

        for (int i = 0; i < y_side; i++)
        {
            for (int j = 0; j < x_side; j++)
            {
                if (exchange_type == "diagonals")
                {
                    if (check_coords(lu_y + i, x_side + lu_x + j) &&
check_coords(y_side + lu_y + i, lu_x + j))
                    {
                        set_pixel(arr[lu_y  +  i][x_side  +  lu_x  +  j],
forth_part[i][j]);
                    }
                    if (check_coords(y_side + lu_y + i, x_side + lu_x +
j) && check_coords(lu_y + i, lu_x + j))
                    {
                        set_pixel(arr[y_side + lu_y + i][x_side + lu_x +
j], first_part[i][j]);
                    }
                    if (check_coords(y_side + lu_y + i, lu_x + j) &&
check_coords(lu_y + i, x_side + lu_x + j))
                    {
```

```
                        set_pixel(arr[y_side + lu_y + i][lu_x + j],
second_part[i][j]);
                    }
                    if (check_coords(lu_y + i, lu_x + j) &&
check_coords(y_side + lu_y + i, x_side + lu_x + j))
                    {
                        set_pixel(arr[lu_y + i][lu_x + j],
third_part[i][j]);
                    }
                }
                if (exchange_type == "clockwise")
                {
                    if (check_coords(lu_y + i, x_side + lu_x + j) &&
check_coords(lu_y + i, lu_x + j))
                    {
                        set_pixel(arr[lu_y + i][x_side + lu_x + j],
first_part[i][j]);
                    }
                    if (check_coords(y_side + lu_y + i, x_side + lu_x +
j) && check_coords(lu_y + i, x_side + lu_x + j))
                    {
                        set_pixel(arr[y_side + lu_y + i][x_side + lu_x +
j], second_part[i][j]);
                    }
                    if (check_coords(y_side + lu_y + i, lu_x + j) &&
check_coords(y_side + lu_y + i, x_side + lu_x + j))
                    {
                        set_pixel(arr[y_side + lu_y + i][lu_x + j],
third_part[i][j]);
                    }
                    if (check_coords(lu_y + i, lu_x + j) &&
check_coords(y_side + lu_y + i, lu_x + j))
                    {
                        set_pixel(arr[lu_y + i][lu_x + j],
forth_part[i][j]);
                    }
                }
                if (exchange_type == "counterclockwise")
                {
                    if (check_coords(lu_y + i, x_side + lu_x + j) &&
check_coords(y_side + lu_y + i, x_side + lu_x + j))
                    {
                        set_pixel(arr[lu_y + i][x_side + lu_x + j],
third_part[i][j]);
```

```
                    }
                    if (check_coords(y_side + lu_y + i, x_side + lu_x +
j) && check_coords(y_side + lu_y + i, lu_x + j))
                    {
                        set_pixel(arr[y_side + lu_y + i][x_side + lu_x +
j], forth_part[i][j]);
                    }
                    if (check_coords(y_side + lu_y + i, lu_x + j) &&
check_coords(lu_y + i, lu_x + j))
                    {
                        set_pixel(arr[y_side + lu_y + i][lu_x + j],
first_part[i][j]);
                    }
                    if (check_coords(lu_y + i, lu_x + j) &&
check_coords(lu_y + i, x_side + lu_x + j))
                    {
                        set_pixel(arr[lu_y + i][lu_x + j],
second_part[i][j]);
                    }
                }
            }
        }
    }

    Rgb Image::get_frequent()
    {

        unordered_map<Rgb, int> color_count;

        for (int i = 0; i < H; i++)
        {
            for (int j = 0; j < W; j++)
            {
                color_count[arr[i][j]]++;
            }
        }

        Rgb frequent_color = arr[0][0];
        int max_count = 1;
        for (const auto &entry : color_count)
        {
            if (entry.second > max_count)
            {
                max_count = entry.second;
```

```
            frequent_color = entry.first;
        }
    }
    return frequent_color;
}

void Image::swap_frequent(Rgb new_pix)
{

    Rgb frequent = get_frequent();

    for (int i = 0; i < H; i++)
    {
        for (int j = 0; j < W; j++)
        {
            if (check_coords(i, j) and arr[i][j] == frequent)
            {
                set_pixel(arr[i][j], new_pix);
            }
        }
    }
}
```

image_bmp.h

```
#ifndef IMG_BMP_H
#define IMG_BMP_H

#include "structs.h"

class Image
{

public:
    BitmapInfoHeader bmif;
    BitmapFileHeader bmfh;
    unsigned int H;
    unsigned int W;
    Rgb **arr;

    Image(string bmp_name)
    {
        arr = read_bmp(bmp_name);
        H = bmif.height;
        W = bmif.width;
```

```
        }

        ~Image()
        {
            for (int i = 0; i < H; ++i)
            {
                delete[] arr[i];
            }
            delete[] arr;
        }

        void printFileHeader();

        void printInfoHeader();

        void set_pixel(Rgb &pix, Rgb new_pix);

        void swap_pixel(Rgb &pix, Rgb &new_pix);

        void fill_zone(int i, int j, Rgb border);

        void plort_circle(int x, int y, int x0, int y0, Rgb new_pix, int
thickness);

        // Модифицированный пример из книги Г.Шилдта «Си для
профессиональных программистов» (алгоритм мичнера)
        void draw_circle(int y0, int x0, int radius, Rgb new_pix, bool
isfill = false, int thickness = 0, bool fillcolor = false, Rgb new_in_pix
= {0, 0, 0});

        void draw_line(int x1, int y1, int x2, int y2, Rgb &new_pix, int
thickness = 0);

        void rgbfilter(string comp_n, int comp_v);

        void draw_square(int h, int w, int side_size, Rgb new_pix, int
thickness = 1, bool isfill = false, Rgb new_in_pix = {0, 0, 0});

        void exchange(int left_up[], int right_down[], string
exchange_type);

        Rgb get_frequent();

        void swap_frequent(Rgb new_pix);
```

```
        Rgb **read_bmp(string bmp_name);

        void write_bmp(string bmp_name);

    private:
        bool check_coords(int y, int x) { return !(y < 0 || y >= H || x
< 0 || x >= W); }
    };

    #endif
```

structs.h

```
    #ifndef STRUCTS_H
    #define STRUCTS_H

    #include <iostream>
    #include <unordered_map>
    #include <vector>
    #include <complex>
    #include <getopt.h>

    using std::cin;
    using std::cout;
    using std::dec;
    using std::endl;
    using std::hex;
    using std::stoi;
    using std::string;
    using std::unordered_map;

    #pragma pack(push, 1)
    struct BitmapFileHeader
    {
        unsigned short signature;
        unsigned int filesize;
        unsigned short reserved1;
        unsigned short reserved2;
        unsigned int pixelArrOffset;
    };

    struct BitmapInfoHeader
    {
        unsigned int headerSize;
```

```
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bitsPerPixel;
    unsigned int compression;
    unsigned int imageSize;
    unsigned int xPixelsPerMeter;
    unsigned int yPixelsPerMeter;
    unsigned int colorsInColorTable;
    unsigned int importantColorCount;
};

struct Rgb
{
    unsigned char b;
    unsigned char g;
    unsigned char r;

    size_t hash() const
    {
        return r ^ (g << 8) ^ (b << 16);
    }

    bool operator==(const Rgb &other) const
    {
        return r == other.r && g == other.g && b == other.b;
    }
};

#pragma pack(pop)

namespace std
{
    // Специализация std::hash для Rgb
    template <>
    struct hash<Rgb>
    {
        size_t operator()(const Rgb &color) const
        {
            return color.hash();
        }
    };
}
```

```
struct Config
{
      string name_out;
      string name_in;
      int name_out_in;
      int name_in_in;
      int info;

      int rgb_filter;
      string rgb_name;
      int rgb_name_in;
      int rgb_val;
      int rgb_val_in;

      int square;
      int sq_lu[2];
      int sq_lu_in;
      int sq_side;
      int sq_side_in;
      int sq_thickness;
      int sq_thickness_in;
      Rgb sq_color;
      int sq_color_in;
      int sq_fill;
      Rgb sq_fill_color;
      int sq_fill_color_in;

      int exchange;
      int ex_lu[2];
      int ex_lu_in;
      int ex_rd[2];
      int ex_rd_in;
      string ex_type;
      int ex_type_in;

      int frequent;
      Rgb freq_color;
      int freq_color_in;
};

#endif
```

Makefile

```
CXX = g++
```

```
CXXFLAGS = -std=c++11

all: main

main: image_io.o image_processing.o input.o main.o
	$(CXX) $(CXXFLAGS) -o cw $^

image_io.o: image_io.cpp structs.h image_bmp.h
	$(CXX) $(CXXFLAGS) -c $<

image_processing.o: image_processing.cpp structs.h image_bmp.h
	$(CXX) $(CXXFLAGS) -c $<

input.o: input.cpp input.h structs.h
	$(CXX) $(CXXFLAGS) -c $<

main.o: main.cpp structs.h image_bmp.h input.h
	$(CXX) $(CXXFLAGS) -c $<

clean:
	rm -f *.o main
```