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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP изображения

Студент гр. 3343 _____ Иванов П.Д.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Иванов Павел

Группа: 3343

Тема: Обработка BMP изображения

Условие (вариант 4.10):

Программа должна иметь следующую функции по обработке изображений:

(1) Заменяет все пиксели одного заданного цвета на другой цвет. Флаг для выполнения данной операции: `--color_replace`. Функционал определяется:

Цвет, который требуется заменить. Флаг `--old_color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--old_color 255.0.0` задаёт красный цвет)

Цвет на который требуется заменить. Флаг `--new_color` (работает аналогично флагу `--old_color`)

(2) Сделать рамку в виде узора. Флаг для выполнения данной операции: `--ornament`. Рамка определяется:

Узором. Флаг `--pattern`. Обязательные значения: rectangle и circle, semicircles. Также можно добавить свои узоры (красивый узор можно получить используя фракталы).

Цветом. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

Шириной. Флаг `--thickness`. На вход принимает число больше 0

Количеством. Флаг `--count`. На вход принимает число больше 0

При необходимости можно добавить дополнительные флаги для необозначенных узоров

(3) Поиск всех залитых прямоугольников заданного цвета. Флаг для выполнения данной операции: `--filled_rects`. Требуется найти все

прямоугольники заданного цвета и обвести их линией. Функционал определяется:

Цветом искомых прямоугольников. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

Цветом линии для обводки. Флаг `--border_color` (работает аналогично флагу `--color`)

Толщиной линии для обводки. Флаг `--thickness`. На вход принимает число больше 0

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 22.05.2024

Дата защиты реферата: 22.05.2024

АННОТАЦИЯ

В ходе курсовой работы реализована программа, осуществляющая обработку изображений в формате BMP. Для взаимодействия с программой предусмотрен интерфейс командной строки (CLI). Программа реализует следующие функции:

Замена цвета: (заменяет все пиксели одного заданного цвета на другой цвет).

Флаг: --color_replace

Создание узорной рамки (создает рамку в виде заданного узора).

Флаг: --ornament

Поиск и выделение залитых прямоугольников (находит все залитые прямоугольники заданного цвета и обводит их линией).

Флаг: --filled_rects

Все функции реализованы в виде отдельных функций с соответствующим вводом/выводом. Сборка проекта осуществляется с помощью утилиты make.

ВВЕДЕНИЕ

Цель работы:

Создание интерактивного консольного приложения для манипуляций с изображениями в формате BMP. Приложение предоставит пользователю следующие возможности:

- Загрузка и сохранение изображений в формате BMP.
- Редактирование изображений.
- Визуализация результатов обработки изображений.

ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

Программа была реализована на языке C++. Код программы разбит на несколько файлов:

- *main.cpp* – содержит функцию main, в которой вызывается функция для обработки аргументов командной строки.
- *cli_parse.cpp* – содержит одноименную функцию, которая обрабатывает аргументы командной строки, функцию *get_colors()* для получения составляющих цвета из входной строки, функцию *print_info()*, которая выводит информацию об изображении.
- *color_replace.cpp* – содержит одноименную функцию, которая заменяет определенный цвет в изображении на новый.
- *find_rectangle.cpp* – содержит одноименную функцию, которая для каждого пикселя подходящего цвета вызывает функцию *check_rectangle()* для проверки на то, является ли фигура, содержащая найденный пиксель прямоугольник (проверяется только рамка прямоугольника). Функция *check_inside()* проверяет совпадают ли все цвета пикселей внутри прямоугольника с заданным цветом. Для обводки найденного прямоугольника используется функция *draw_frame()*.
- *get_pixel.cpp* – содержит одноименную функцию, которая является служебной для остальных и используется для получения пикселя по переданным координатам (координаты используются как при навигации по двумерному массиву).
- *ornament.cpp* – содержит одноименную функцию, которая в зависимости от переданной строки вызывает нужную функцию согласно заданию. Функции *draw_semicircle()* и *draw()* используются для рисования рамки из полукругов. Функции *draw_rectangle()* и *draw_long_line()* используются для рисования рамки из прямоугольников. Функция *draw_circle()* используется для рисования круговой рамки

- *read_file.cpp / write_file.cpp* – содержат функции для чтения и записи изображения соответственно.
- *functions.h* – заголовочный файл, который содержит объявления некоторых функций для того, чтобы они могли вызываться в других .cpp файлах проекта.
- *data_types.h* – заголовочный файл, который содержит определения структур BMPHeader, PICInfo, OptionTools, long_flags, а также в этом файле содержатся директивы include для подключения библиотек.

ТЕСТИРОВАНИЕ

Рисунок 1 — исходное изображение

1. Тестирование рамки с узором из полукругов: *--input ./images/big.bmp --thickness 13 --ornament --color 255.200.20 --count 4 --pattern semicircles*

Рисунок 2 — работа функции ornament(semicircles)

2. Тестирование замены цвета: *--input ./images/big.bmp --color_replace --old_color 99.198.247 --new_color 105.47.200*

Рисунок 3 — работа функции color_replace

3. Тестирование поиска прямоугольников: *--input ./images/big.bmp --filled_rects --color 99.198.247 --border_color 255.200.20 --thickness 10*

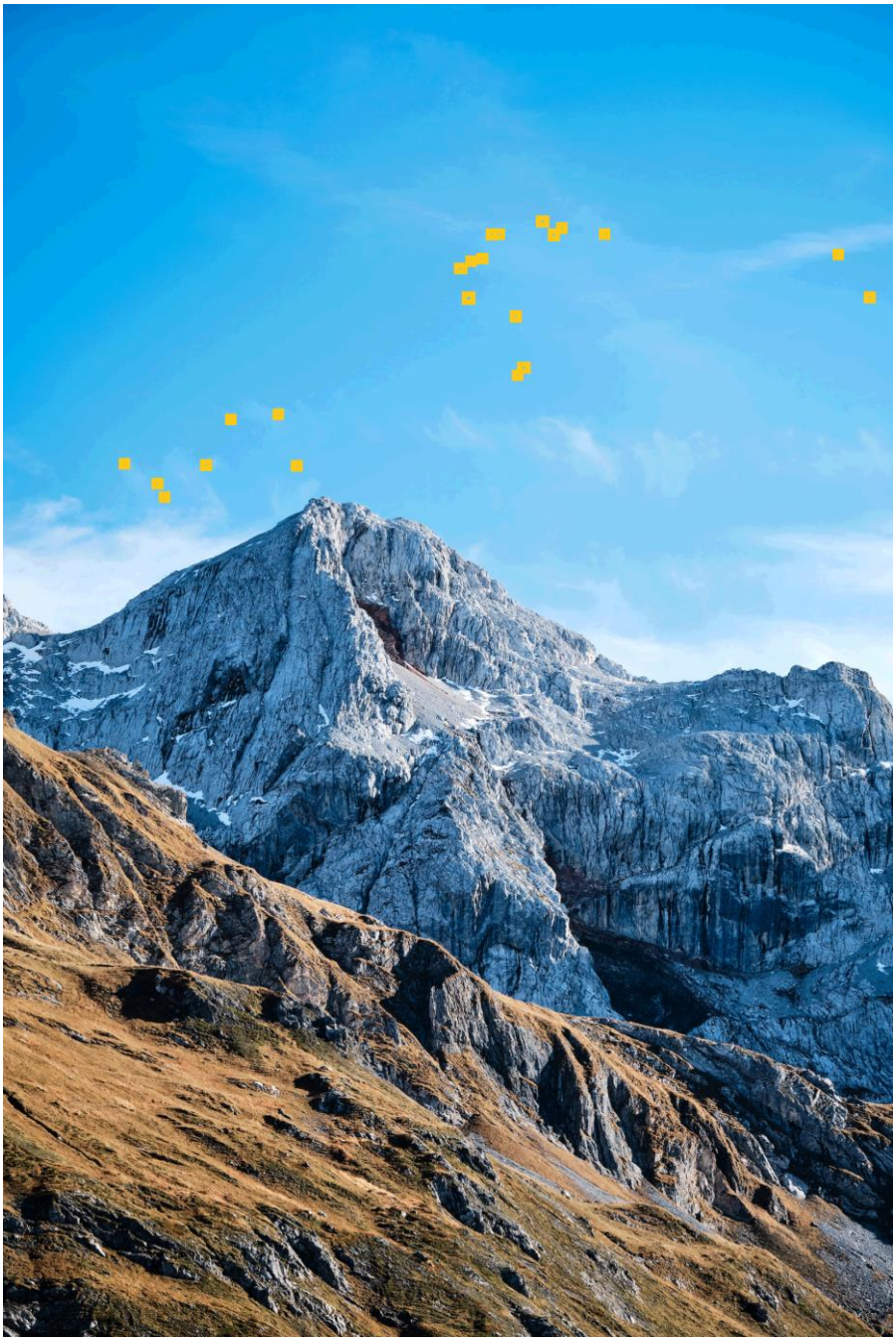

Рисунок 4 — работа функции find_rectangle

ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы было создано приложение на языке программирования C++ для манипуляций с изображениями в формате BMP. Предоставляемый функционал программы может быть выбран пользователем через командную строку. Сборка приложения осуществляется при помощи инструмента make. После сборки приложение запускается из командной строки, где пользователь может выбрать одну или несколько из поддерживаемых функций для обработки изображения.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp

```
#include "../include/functions.h"

int main(int argc, char* argv[]){
    cli_parse(argc, argv);
}
```

cli_parse.cpp

```
#include "../include/functions.h"

void print_info(PICInfo& info, const string& file_name){
    cout << "File name: " << file_name << endl;
    cout << "File type: " << info.header.Type[0] << info.header.Type[1] << endl;
    cout << "Height: " << info.header.Height << endl;
    cout << "Width: " << info.header.Width << endl;
    cout << "Size: " << info.header.Size << " bytes" << endl;
    cout << "Color depth: " << info.header.BitCount << endl;
    exit(0);
}

int* get_colors(string& colors){
    istringstream stream_colors(colors);
    int* arr_colors = new int[3];
    char dot1, dot2;

    try{
        if (!(stream_colors >> arr_colors[0] >> dot1 >> arr_colors[1] >> dot2 >>
arr_colors[2]) or dot1 != '.' or dot2 != '.') {
```

```
            throw 1;
        }
        for (int i = 0; i < 3; ++i) {
            if (arr_colors[i] < 0 or arr_colors[i] > 255) {
                throw 1;
            }
        }
    }catch (int e){
        cerr << "Colors entered incorrectly: " << colors << endl;
        exit(40);
    }

    return arr_colors;
}

void cli_parse(int argc, char* argv[]){
    int opt;
    PICInfo info{};
    OptionTools tools{};
    deque<function<void()>> operations;
    vector<char> used_flags;
    string output_file_name = "out.bmp";
    string input_file_name;
    string current_func;

    while((opt = getopt_long(argc, argv, "i:o:hrnf", long_flags, nullptr)) != -1){
        switch(opt){
            case 'h': {
                cout << HELP_INFO << endl;
                exit(0);
```

```
            }
            case 'i': {
                input_file_name = optarg;
                break;
            }
            case 'o': {
                output_file_name = optarg;
                break;
            }
            case 256: {
                auto get_info = [&input_file_name, &info] {print_info(info, input_file_name);};
                operations.emplace_back(get_info);
                break;
            }
            case 'r': {
                used_flags.push_back('r');
                break;
            }
            case 257: {
                string s_old_colors = optarg;
                tools.old_color = get_colors(s_old_colors);
                break;
            }
            case 258: {
                string s_new_colors = optarg;
                tools.new_color = get_colors(s_new_colors);
                break;
            }
            case 'n': {
```

```
    used_flags.push_back('n');
    current_func = "ornament";
    break;
}
case 259: {
    tools.pattern = optarg;
    break;
}
case 260: {
    string s_colors = optarg;
    if(current_func == "ornament"){
        tools.pattern_color = get_colors(s_colors);
    } else if (current_func == "find_rects"){
        tools.rect_color = get_colors(s_colors);
    } else {
        for (int i = 0; i < argc; ++i) {
            if(strcmp(argv[i], "--ornament") == 0){
                tools.pattern_color = get_colors(s_colors);
            }
            if(strcmp(argv[i], "--filled_rects") == 0){
                tools.rect_color = get_colors(s_colors);
            }
        }
    }
    break;
}
case 261: {
    try {
        int value = stoi(optarg);
        if(current_func == "ornament"){
```

```
            tools.pattern_thickness = value;
            if(tools.pattern_thickness < 1){
                throw invalid_argument("thickness less than 1");
            }
        } else if (current_func == "find_rects"){
            tools.border_thickness = value;
            if(tools.border_thickness < 1){
                throw invalid_argument("thickness less than 1");
            }
        } else {
            for (int i = 0; i < argc; ++i) {
                if(strcmp(argv[i], "--ornament") == 0){
                    tools.pattern_thickness = value;
                }
                if(strcmp(argv[i], "--filled_rects") == 0){
                    tools.border_thickness = value;
                }
            }
        }
    } catch (invalid_argument& e) {
        cerr << "Invalid argument for thickness parameter" << endl;
        exit(40);
    }
    break;
}
case 262: {
    try {
        tools.count = stoi(optarg);
        if(tools.count < 1){
            throw invalid_argument("count less than 1");
```

```
            }
        } catch (invalid_argument& e) {
            cerr << "Invalid argument for count parameter" << endl;
            exit(40);
        }
        break;
    }
    case 'f': {
        used_flags.push_back('f');
        current_func = "find_rects";
        break;
    }
    case 263: {
        string s_border_colors = optarg;
        int* color = get_colors(s_border_colors);
        tools.border_color = color;
        break;
    }
    default:
        cerr << "Unknown flag" << endl;
        exit(40);
  }
}
if(input_file_name.empty()){
  input_file_name = argv[argc - 1];
}

if(input_file_name == output_file_name){
  cerr << "The names of the input and output files are the same" << endl;
  exit(40);
```

```
    }

    auto read_file = [input_file_name, &info] {info = read(input_file_name);};
    operations.emplace_front(read_file);

    for(char flag : used_flags) {
       if (flag == 'r') {
          auto replace = [&info, &tools] {
             color_replace(info, tools.old_color, tools.new_color);
          };
          operations.emplace_back(replace);
       }
       if (flag == 'f') {
          auto find = [&info, &tools] {
             find_rectangle(info, tools.rect_color, tools.border_thickness,
tools.border_color);
          };
          operations.emplace_back(find);
       }
       if (flag == 'n') {
          auto frame = [&info, &tools] {
             ornament(info, tools.pattern, tools.pattern_color,
tools.pattern_thickness, tools.count);
          };
          operations.emplace_back(frame);
       }
    }

    auto write_file = [output_file_name, &info] { write(info,
output_file_name);};
```

```
        operations.emplace_back(write_file);


        while(!operations.empty()){
            operations.front()();
            operations.pop_front();
        }

        delete tools.old_color;
        delete tools.new_color;
        delete tools.pattern_color;
        delete tools.rect_color;
        delete tools.border_color;
    }

    color_replace.cpp
    #include "../include/functions.h"

    void color_replace(PICInfo& info, int* old_colors, int* new_colors) {
        int old_red = old_colors[0];
        int old_green = old_colors[1];
        int old_blue = old_colors[2];
        int new_red = new_colors[0];
        int new_green = new_colors[1];
        int new_blue = new_colors[2];


        for (unsigned int i = 0; i < size_image - 2; i+=3) {
            if((info.pixels[i] == old_blue) && (info.pixels[i+1] == old_green) &&
(info.pixels[i+2] == old_red)){
```

```
            info.pixels[i] = new_blue;
            info.pixels[i+1] = new_green;
            info.pixels[i+2] = new_red;
        }
    }
}
```

find_rectangle.cpp

```
#include "../include/functions.h"

int* size_of_rectangle = new int[2];

void draw_frame(int rw, int col, int height, int length, int thickness, PICInfo& info, int* border_color){
    int x0 = rw - thickness;
    int y0 = col - thickness;
    int x1 = rw + height + thickness;
    int y1 = col + length + thickness;
    int bord_red = border_color[0];
    int bord_green = border_color[1];
    int bord_blue = border_color[2];

    for(int x = x0; x < x1; ++x){
        for(int y = y0; y < y1; ++y){
            if(x < 0 or y < 0 or x >= info.header.Height or y >= info.header.Width){
                continue;
            }
            if((x < rw or (x >= rw + height)) or (y < col or (y >= col + length))){
                Pixel px = get_pixel(info, x, y);
                *px.red = bord_red;
```

```
                *px.green = bord_green;
                *px.blue = bord_blue;
            }
        }
    }
}

bool check_inside(int row_start, int column_start, int row_stop, int column_stop, int red, int green, int blue, PICInfo& info){
    for (int row = row_start; row <= row_stop; ++row){
        for (int column = column_start; column <= column_stop; ++column) {
            Pixel px = get_pixel(info, row, column);
            if(*px.blue != blue or *px.green != green or *px.red != red) {
                return false;
            }
        }
    }
    return true;
}

bool check_rectangle(PICInfo& info, int row, int column, int red, int green, int blue){
    if(row == info.header.Height - 1 or column == info.header.Width - 1){
        return false;
    }
    if(row != 0) {
        Pixel top_px = get_pixel(info, row - 1, column);
        if(*top_px.red == red and *top_px.green == green and *top_px.blue == blue){
            return false;
```

```
            }
        }
        Pixel bottom_px{};
        Pixel next_frame_px{};
        Pixel diagonal_top_px{};
        int length = 0;
        int height = 0;
        int temp_row = row;
        int temp_column = column;

        //обход верхней стороны
        do{
            if(temp_column == info.header.Width - 1){
                length++;
                break;
            }
            bottom_px = get_pixel(info, row + 1, temp_column);
            next_frame_px = get_pixel(info, row, temp_column + 1);
            if(row != 0){
                diagonal_top_px = get_pixel(info, row - 1, temp_column + 1);
                if (*diagonal_top_px.red == red and *diagonal_top_px.green == green
and *diagonal_top_px.blue == blue) {
                    return false;
                }
            }
            if(*bottom_px.red != red or *bottom_px.green != green or
*bottom_px.blue != blue){
                return false;
            }
            length++;
```

```
            temp_column++;
        }while(*next_frame_px.blue == blue and *next_frame_px.green == green and *next_frame_px.red == red);
        if(length <= 1){
            return false;
        }
        //левый диагональный пиксель
        if(row != 0 and column != 0){
            Pixel left_diagonal = get_pixel(info, row - 1, column - 1);
            if(*left_diagonal.red == red and *left_diagonal.green == green and *left_diagonal.blue == blue){
                return false;
            }
        }

        //обход левой стороны
        if(column != 0) {
            Pixel top_px = get_pixel(info, row, column - 1);
            if(*top_px.red == red and *top_px.green == green and *top_px.blue == blue){
                return false;
            }
        }
        do{
            if(temp_row == info.header.Height - 1){
                height++;
                break;
            }
            bottom_px = get_pixel(info, temp_row, column + 1);
            next_frame_px = get_pixel(info, temp_row + 1, column);
```

```
            if(column != 0) {
                diagonal_top_px = get_pixel(info, temp_row + 1, column - 1);
                if(*diagonal_top_px.red == red and *diagonal_top_px.green == green
and *diagonal_top_px.blue == blue){
                    return false;
                }
            }
            if(*bottom_px.red != red or *bottom_px.green != green or
*bottom_px.blue != blue){
                return false;
            }
            height++;
            temp_row++;
        }while(*next_frame_px.red == red and *next_frame_px.green == green and
*next_frame_px.blue == blue);
        if(height <= 1){
            return false;
        }

        //обход нижней стороны
        temp_row = row + height - 1;
        temp_column = column;
        int down_length = 0;
        Pixel top_px{};
        Pixel diagonal_down_px{};

        if(temp_row != info.header.Height - 1){
            bottom_px = get_pixel(info, temp_row + 1, column);
            if (*bottom_px.red == red and *bottom_px.green == green and
*bottom_px.blue == blue) {
```

```
            return false;
        }
    }
    do{
        if(temp_column == info.header.Width - 1){
            down_length++;
            break;
        }
        top_px = get_pixel(info, temp_row - 1, temp_column);
        next_frame_px = get_pixel(info, temp_row, temp_column + 1);
        if(temp_row != info.header.Height - 1){
            diagonal_down_px = get_pixel(info, temp_row + 1, temp_column + 1);
            if (*diagonal_down_px.red == red and *diagonal_down_px.green == green and *diagonal_down_px.blue == blue) {
                return false;
            }
        }
        if(*top_px.red != red or *top_px.green != green or *top_px.blue != blue){
            return false;
        }
        down_length++;
        temp_column++;
    }while(*next_frame_px.blue == blue and *next_frame_px.green == green and *next_frame_px.red == red);
    if(down_length != length){
        return false;
    }

    //обход правой стороны
    temp_row = row;
```

```
        temp_column = column + length - 1;
        int right_height = 0;

        if(temp_column != info.header.Width - 1){
            top_px = get_pixel(info, row, temp_column + 1);
            if (*top_px.red == red and *top_px.green == green and *top_px.blue ==
blue){
                return false;
            }
        }
        do{
            if(temp_row == info.header.Height - 1){
                right_height++;
                break;
            }
            bottom_px = get_pixel(info, temp_row, temp_column - 1);
            next_frame_px = get_pixel(info, temp_row + 1, temp_column);
            if(temp_column != info.header.Width - 1) {
                diagonal_top_px = get_pixel(info, temp_row + 1, temp_column + 1);
                if (*diagonal_top_px.red == red and *diagonal_top_px.green == green
and *diagonal_top_px.blue == blue) {
                    return false;
                }
            }
            if(*bottom_px.red != red or *bottom_px.green != green or
*bottom_px.blue != blue){
                return false;
            }
            right_height++;
            temp_row++;
```

```
        }while(*next_frame_px.red == red and *next_frame_px.green == green and
*next_frame_px.blue == blue);
        if(right_height != height){
            return false;
        }

        //проверка пикселей внутри прямоугольника
        if(!check_inside(row, column, row + height - 1, column + length - 1, red,
green, blue, info)){
            return false;
        }

        size_of_rectangle[0] = height;
        size_of_rectangle[1] = length;

        return true;
    }

    void find_rectangle(PICInfo& info, int* rect_color, int thickness, int*
border_color){
        int rect_red = rect_color[0];
        int rect_green = rect_color[1];
        int rect_blue = rect_color[2];
        for(int rw = 0; rw < info.header.Height; ++rw){
            for(int col = 0; col < info.header.Width; ++col){
                Pixel px = get_pixel(info, rw, col);
                if(*px.blue == rect_blue and *px.green == rect_green and *px.red ==
rect_red){
                    if(check_rectangle(info, rw, col, rect_red, rect_green, rect_blue)){
```

```
                draw_frame(rw, col, size_of_rectangle[0], size_of_rectangle[1], thickness, info, border_color);
            }
        }
    }
}
```

get_pixel.cpp

```
#include "../include/functions.h"

Pixel get_pixel(PICInfo& info, int row, int column){
    int bytesPerPixel = info.header.BitCount / 8;
    int bytesPerRow = (bytesPerPixel * info.header.Width + 3) & ~3;
    int index = ((info.header.Height - 1 - row) * bytesPerRow) + (column * bytesPerPixel);
    return Pixel{&(info.pixels.at(index + 2)), &(info.pixels.at(index + 1)), &(info.pixels.at(index))};
}
```

ornament.cpp

```
#include "../include/functions.h"

void draw_circle(PICInfo& info, int red, int green, int blue){
    int center_row = info.header.Height/2;
    int center_col = info.header.Width/2;
    int radius = min(info.header.Height/2, info.header.Width/2);

    for(int rw = 0; rw < info.header.Height; ++rw){
        for(int col = 0; col < info.header.Width; ++col){
```

```
            int k1 = abs(center_row - rw);
            int k2 = abs(center_col - col);
            if((hypot(k1, k2)) > radius){
               Pixel px = get_pixel(info, rw, col);
               *px.blue = blue;
               *px.green = green;
               *px.red = red;
            }
         }
      }
   }

   void draw_long_line(PICInfo& info, int thickness, int row, int column, int red, int green, int blue){
      for(int rw = row; rw < row+thickness; ++rw){
         for(int col = column; col < info.header.Width - column; ++col){
            Pixel px = get_pixel(info, rw, col);
            *px.blue = blue;
            *px.green = green;
            *px.red = red;
         }
      }
   }

   void draw_rectangle(PICInfo& info, int thickness, int row, int column, int red, int green, int blue){
      draw_long_line(info, thickness, row, column, red, green, blue);
      for(int rw = row+thickness; rw < info.header.Height - row; ++rw){
         for(int col = column; col < column+thickness; ++col){
            Pixel px = get_pixel(info, rw, col);
```

```
                *px.blue = blue;
                *px.green = green;
                *px.red = red;
            }
            for(int col = info.header.Width - row - thickness; col < info.header.Width - row; ++col){
                Pixel px = get_pixel(info, rw, col);
                *px.blue = blue;
                *px.green = green;
                *px.red = red;
            }
        }
        draw_long_line(info, thickness, info.header.Height - row - thickness, column, red, green, blue);
    }

    void draw(PICInfo& info, int center_x, int center_y, int thickness, int radius, int red, int green, int blue, const string& dim){
        int radius_square = (int)pow(radius, 2);
        int outer_radius_square = (int)pow(radius + thickness, 2);

        if (dim == "up" or dim == "down"){
            int start = (dim == "up") ? 0 : info.header.Height - 1;
            int stop = (dim == "up") ? radius + thickness : info.header.Height - radius - thickness;
            int step = (dim == "up") ? 1 : -1;
            for(int rw = start; (dim == "up" && rw <= stop) || (dim == "down" && rw >= stop); rw+=step){
                for(int col = 0; col < info.header.Width; ++col){
                    int distance_x = abs(center_x - rw);
```

```
                int distance_y = abs(center_y - col);

                bool greater_than_radius = (distance_x * distance_x) + (distance_y * distance_y) >= radius_square;
                bool less_than_outer_radius = (distance_x * distance_x) + (distance_y * distance_y) < outer_radius_square;
                if (greater_than_radius && less_than_outer_radius){
                    Pixel px = get_pixel(info, rw, col);
                    *px.blue = blue;
                    *px.green = green;
                    *px.red = red;
                }
            }
        }
    }
    if (dim == "right" or dim == "left"){
        int start = (dim == "left") ? 0 : info.header.Width - 1;
        int stop = (dim == "left") ? radius + thickness : info.header.Width - radius - thickness;
        int step = (dim == "left") ? 1 : -1;
        for(int rw = 0; rw < info.header.Height; ++rw){
            for(int col = start; (dim == "left" && col <= stop) || (dim == "right" && col >= stop); col+=step){
                int distance_x = abs(center_x - rw);
                int distance_y = abs(center_y - col);

                bool greater_than_radius = (distance_x * distance_x) + (distance_y * distance_y) >= radius_square;
                bool less_than_outer_radius = (distance_x * distance_x) + (distance_y * distance_y) < outer_radius_square;
```

```
                if (greater_than_radius && less_than_outer_radius){
                    Pixel px = get_pixel(info, rw, col);
                    *px.blue = blue;
                    *px.green = green;
                    *px.red = red;
                }
            }
        }
    }
}

void draw_semicircle(PICInfo& info, int thickness, int count, int red, int green, int blue){
    double temp = ((double)info.header.Width / ((double)count * 2)) - ((double)thickness / 2);
    int horizontal_radius;
    if((info.header.Width % (count*2) != 0) or (thickness % 2 != 0)){
        horizontal_radius = (int)ceil(temp);
    } else {
        horizontal_radius = (int)temp;
    }
    int step = 2 * horizontal_radius + thickness - 1;
    int start = (int)ceil((double)thickness / 2) + horizontal_radius;

    //для ситуаций типа s 255 255 255 4 10 (500x210) при радиусе 23 - останется место, при 24 - поместятся не все
    if((start + ((count - 1) * step) + horizontal_radius + thickness) < info.header.Width){
        horizontal_radius += 1;
        step = 2 * horizontal_radius + thickness - 1;
```

```
            start = (int)ceil((double)thickness / 2) + horizontal_radius;
        }

        int column = start;
        for (int i = 0; i < count; ++i) {
            draw(info, 0, column, thickness, horizontal_radius, red, green, blue, "up");
            draw(info, info.header.Height-1, column, thickness, horizontal_radius, red, green, blue, "down");
            column += step;
        }


        double temp_vertical = ((double)info.header.Height / ((double)count * 2)) - ((double)thickness / 2);
        int vertical_radius;
        if((info.header.Height % (count*2) != 0) or (thickness % 2 != 0)){
            vertical_radius = (int)ceil(temp_vertical);
        } else {
            vertical_radius = (int)temp_vertical;
        }
        int step_vertical = 2 * vertical_radius + thickness - 1;
        int start_vertical = (int)ceil((double)thickness / 2) + vertical_radius;

        if((start_vertical + ((count - 1) * step_vertical) + vertical_radius + thickness) < info.header.Height){
            vertical_radius += 1;
            step_vertical = 2 * vertical_radius + thickness - 1;
            start_vertical = (int)ceil((double)thickness / 2) + vertical_radius;
        }
```

```
        int row = start_vertical;
        for (int i = 0; i < count; ++i) {
            draw(info, row, 0, thickness, vertical_radius, red, green, blue, "left");
            draw(info, row, info.header.Width-1, thickness, vertical_radius, red, green, blue, "right");
            row += step_vertical;
        }
    }


    void ornament(PICInfo& info, string& type, int* colors, int thickness, int count) {
        int red = colors[0];
        int green = colors[1];
        int blue = colors[2];

        if (type == "rectangle"){
            int row = 0, column = 0;
            for (int i = 0; i < count; ++i) {
                draw_rectangle(info, thickness, row, column, red, green, blue);
                row += 2*thickness;
                column += 2*thickness;
            }
        }else if (type == "circle"){
            draw_circle(info, red, green, blue);
        }else if (type == "semicircles"){
            draw_semicircle(info, thickness, count, red, green, blue);
        }else{
            cerr << "Incorrect type of ornament" << endl;
            exit(40);
```

```
    }

}

read_file.cpp
#include "../include/functions.h"
unsigned int size_image;

PICInfo read(const string& file_name){
    BMPHeader header{};

    ifstream file_in(file_name, ios::binary);
    if(!file_in.is_open()){
        cerr << "Can't open file" << endl;
        exit(40);
    }

    file_in.read(reinterpret_cast<char*>(&header), sizeof(header));

    if(header.Type[0] != 'B' or header.Type[1] != 'M'){
        cerr << "Incorrect file format" << endl;
        exit(40);
    }
    if(header.BitCount != 24){
        cerr << "Bit per pixel must be equal 24" << endl;
        exit(40);
    }
    if (header.Width <= 0 || header.Height <= 0) {
        cerr << "Incorrect size of image" << endl;
        exit(40);
```

```
    }
    if (header.Compression != 0) {
        cerr << "Compression must be equal 0" << endl;
        exit(40);
    }

    int bytes_per_pixel = header.BitCount / 8;
    int bytes_per_row = ((header.Width * bytes_per_pixel + 3) / 4) * 4;
    size_image = header.Height * bytes_per_row;



    vector<uint8_t> pixels;
    pixels.resize(size_image);
    file_in.seekg(header.OffsetBits, iostream::beg);
    file_in.read(reinterpret_cast<char*>(pixels.data()), size_image);

    PICInfo info{header, pixels};
    file_in.close();

    return info;
}
```

write_file.cpp

```
#include "../include/functions.h"

void write(PICInfo& info, const string& file_name){
    ofstream file_out(file_name, ios::binary);
    if(!file_out){
        cerr << "Can't make file" << endl;
        exit(40);
```

```
    }

    file_out.write(reinterpret_cast<char*>(&info.header), sizeof(info.header));
    file_out.seekp(info.header.OffsetBits, ios::beg);
    file_out.write(reinterpret_cast<char*>(info.pixels.data()), size_image);

    file_out.close();
}
```