**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Обработка изображений на языке Си

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Хайруллов Д.Л. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |
| --- |
| Студент Хайруллов Д.Л. |
| Группа 3342 |
| Тема работы: Обработка изображений на языке Си |

Исходные данные:

Вариант 4.10

|  |
| --- |
| Общие сведения:  24 бита на цвет  без сжатия  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). Подробнее здесь: https://se.moevm.info/doku.php/courses:programming:cw\_spring\_ornament  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 |

|  |  |  |
| --- | --- | --- |
| Содержание пояснительной записки:  Разделы пояснительный записки: «Содержание», «Введение», «Ход работы», «Структуры», «Заключение», «Список использованных источников», «Приложение А. Результаты тестирования», «Приложение Б. Исходный код программы». | | |
| Предполагаемый объем пояснительной записки:  Не менее 37 страниц. | | |
| Дата выдачи задания: 18.03.2024 | | |
| Дата сдачи реферата: 29.05.2024 | | |
| Дата защиты реферата: 29.05.2024 | | |
| Студент |  | Хайруллов Д.Л. |
| Преподаватель |  | Глазунов С.А. |

**Аннотация**

Целью данной курсовой работы является разработка программы для обработки изображений формата bmp на языке программирования С. Программа считывает изображение формата bmp, получает на вход различные опции, в зависимости от которых программа преобразует изображение. В программе используются стандартные библиотеки Си, с помощью которых были реализованы необходимые функции.

**Summary**

The aim of this coursework is to develop a program for processing BMP image files using the C programming language. The program reads BMP image files, takes various options as input, and based on these options, processes the image accordingly. Standard C libraries are used in the program to implement the necessary functions.**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 7 |
| 1. | Функции считывания изображения | 8 |
| 1.1.  1.2.  1.3. | Функция ввода текста  Функция записи изображения  Функция вывода информации об изображении | 8  8  8 |
| 2. | Функции обработки изображения | 10 |
| 2.1. | Функция первого задания | 10 |
| 2.2.  2.3. | Функция второго задания  Функция третьего задания | 10  10 |
| 3. | Остальные функции | 12 |
| 3.1. | Функция закрашивания области | 12 |
| 3.2.  3.3.  3.4.  3.5. | Функция замены цвета  Функция получения цвета  Функция обмена значениями  Функция main | 12  12  12  12 |
|  | Заключение | 14 |
|  | Список использованных источников | 15 |
|  | Приложение А. Результаты тестирования  Приложение Б. Исходный код | 16  21 |

**введение**

Целью данной курсовой работы является разработка программы для обработки изображения формата bmp на языке программирования С.

Основные задачи:

* 1. Считывание изображения
  2. Считывание опций для обработки изображений
  3. Обработка изображения
  4. Сохранения изображения

**1. Функции ввода и вывода**

* 1. **Функция считывания изображения**

Данная функция read\_bmp открывает файл изображения BMP, считывает заголовок файла и информацию о изображении. Затем происходит выделение памяти под данные изображения на основе его высоты и ширины. Последующее чтение данных пикселей изображения происходит в цикле по строкам. Наконец, файл закрывается, и функция возвращает структуру BMP, содержащую данные изображения.

* 1. **Функция записи изображения**

Функция write\_bmp открывает файл для записи в формате BMP, записывает заголовок файла и информацию о изображении из структуры BMP\_file. Затем данные пикселей изображения записываются в файл в цикле по строкам. По завершении записи файл закрывается.

* 1. **Функции вывода информации об изображении**

Функция printFileHeader выводит информацию о заголовке файла BMP, включая сигнатуру, размер файла, резервные поля и смещение массива пикселей.

Функция printInfoHeader выводит информацию о заголовке информации об изображении BMP, включая размер заголовка, ширину и высоту изображения, количество бит на пиксель, метод сжатия, размер изображения, разрешение и дополнительные параметры. Вызов функции и результат ее работы представлены в приложении А на рисунке 3.

* 1. **Функция вывода справки**

Функция printHelp выводит краткую справку о программе для выполнения курсовой работы по опции 4.10, созданной Khairullov Dinar. Выводит список доступных опций с их описанием, включая информацию о том, как использовать каждую опцию: помощь, информация о файле, установка имени входного файла, установка имени выходного файла, замена цвета пикселей, установка старого и нового цветов, рисование орнаментов, установка шаблона орнамента, установка цвета заполненного прямоугольника, установка толщины линий, установка количества элементов орнамента, поиск заполненных прямоугольников заданного цвета и добавление границ и установка цвета границ. Вызов функции и результат ее работы представлены в приложении А на рисунке 2.

**2. Функции обработки ИЗОБРАЖЕНИЯ**

* 1. **Функция первого задания**

Функция color\_replace заменяет все пиксели изображения bmp\_file, имеющие определенный старый цвет old\_color, на новый цвет new\_color. Для этого функция проходит по каждому пикселю изображения и, если его цвет соответствует старому цвету, вызывает функцию color\_pixel, которая заменяет цвет пикселя на новый цвет.. Вызов функции и результат ее работы представлены в приложении А на рисунке 4 и рисунке 5.

* 1. **Функции второго задания**

Функция rectangle\_ornament рисует прямоугольные орнаменты определенной толщины и количества на изображении bmp\_file, используя указанный цвет color. Орнаменты направлены по диагонали. Вызов функции и результат ее работы представлены в приложении А на рисунке 6 и рисунке 7.

Функция circle\_ornament рисует круглый орнамент с центром в центре изображения bmp\_file и радиусом, достигающим до ближайшего края изображения. Орнамент закрашивает пиксели за пределами круга указанным цветом. Вызов функции и результат ее работы представлены в приложении А на рисунке 8 и рисунке 9.

Функция semicircle\_ornament рисует полукруглые орнаменты вертикально и горизонтально на изображении bmp\_file. Орнаменты имеют определенную толщину, радиусы от центра внутренних кругов и количество. Орнаменты закрашиваются указанным цветом. РВызов функции и результат ее работы представлены в приложении А на рисунке 10 и рисунке 11.

* 1. **Функция третьего задания**

Функция filled\_rects предназначена для поиска и закрашивания прямоугольных областей одного цвета на изображении. Вначале функция определяет размеры изображения (H - высота, W - ширина) и начальные значения координат прямоугольника (x\_left, y\_down, x\_right, y\_up). Также выделяется память под массивы start\_colored\_pixels и end\_colored\_pixels, используемые для хранения координат начала и конца прямоугольных областей.

Затем функция перебирает все пиксели изображения и ищет пиксели заданного цвета (color). При нахождении такого пикселя функция определяет границы прямоугольной области, заполненной цветом. Для этого определяются начальные координаты и ширина прямоугольника. Затем осуществляется проверка пикселей в пределах найденного прямоугольника.

В процессе работы функция использует вспомогательную функцию colored\_pixel\_check, обрабатывает флаги и осуществляет динамическое выделение памяти с помощью realloc для массивов хранения информации о прямоугольных областях. Для найденных прямоугольников рисуется рамка внутри фигуры. Вызов функции и результат ее работы представлены в приложении А на рисунке 12 и рисунке 13.

**3. Остальные функции**

* 1. **Функция закрашивания области**

Функция color\_area предназначена для закрашивания прямоугольной области на изображении определенным цветом. В начале функции определяются размеры изображения (H - высота, W - ширина) и проверяется корректность входных координат прямоугольника. Если координаты находятся за пределами изображения, функция завершает свою работу.

Затем функция проверяет правильность порядка координат прямоугольника и, если необходимо, меняет их местами. После этого функция проходит по всем пикселям в границах указанной области и закрашивает их определенным цветом с помощью вспомогательной функции color\_pixel.

* 1. **Функция замены цвета**

Функция color\_replace предназначена для замены определенного цвета на другой на всем изображении. В начале функции определяются размеры изображения (H - высота, W - ширина). Затем функция проходит по всем пикселям изображения, и если цвет текущего пикселя совпадает с цветом old\_color, то этот пиксель заменяется на цвет new\_color с помощью вспомогательной функции color\_pixel.

* 1. **Функция получения цвета**

Функция get\_color преобразует строковое значение цвета в структуру Rgb, представляющую цвет в формате RGB.

* 1. **Функция обмена значениями**

Функция swap выполняет обмен значениями двух целочисленных переменных, переданных по указателям.

* 1. **Функция main**

Функция main представляет точку входа в программу и обрабатывает переданные аргументы командной строки. Она использует библиотеку getopt\_long для разбора опций командной строки и выполнения соответствующих действий в зависимости от переданных параметров.

В этой функции определены различные опции для работы с изображением, такие как замена цвета, добавление узора, вывод информации о изображении и другие операции. Она также обрабатывает ошибки ввода и выводит сообщения об ошибке при необходимости.

После обработки опций командной строки, функция main читает изображение из файла, получает его высоту и ширину, а затем в зависимости от выбранной опции выполняет соответствующие операции над изображением.

**заключение**

При выполнении курсовой работы были изучены основные методы работы с изображениями формата bmp в языке программирования C. Для выполнения заданий работы была написана программа, использующая стандартные библиотеки и функции данного языка программирования. Тестирование программы показало, что все функции программы работают корректно и выполняют необходимые задачи в соответствии с условиями.

Была создана программа, выполняющая все необходимые задачи корректно. Тем самым можно сказать, что все поставленные в работе цели были достигнуты.

В приложении А представлены результаты тестирования программы.

В приложении Б представлен код программы.

**список использованных источников**

1. Курс “Программирование на Си. Практические задания. Второй семестр”. URL <https://e.moevm.info/course/view.php?id=18>

2. Язык программирования C / Керниган Брайан, Ритчи Деннис. СПб.: "Финансы и статистика", 2003.

**приложение А**

**Результаты Тестирования**

Изначальное изображение



Рисунок 1 – начальное изображение

Тест 1 – вывод справки

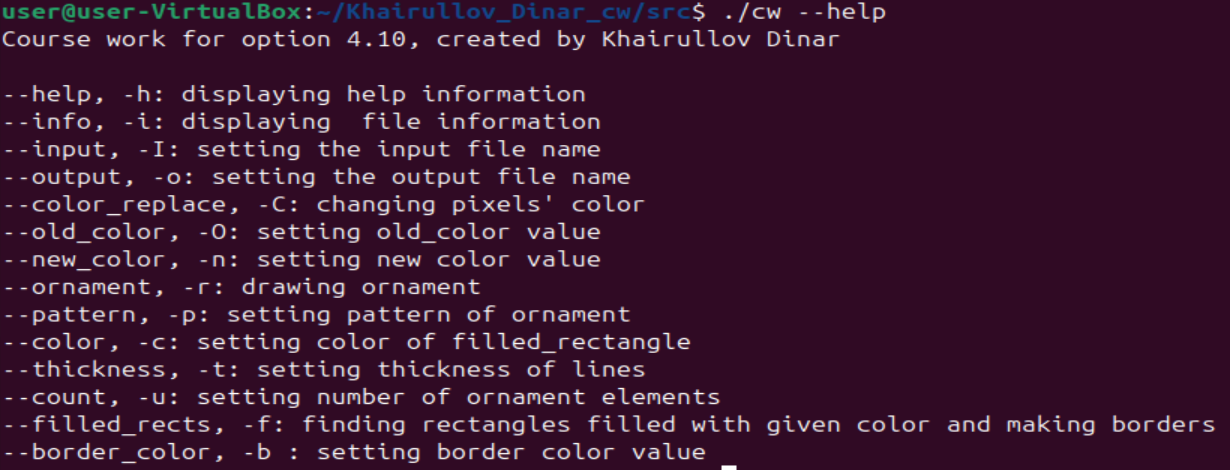


Рисунок 2 – вывод справки в консоль

Тест 2 – вывод информации об изображении

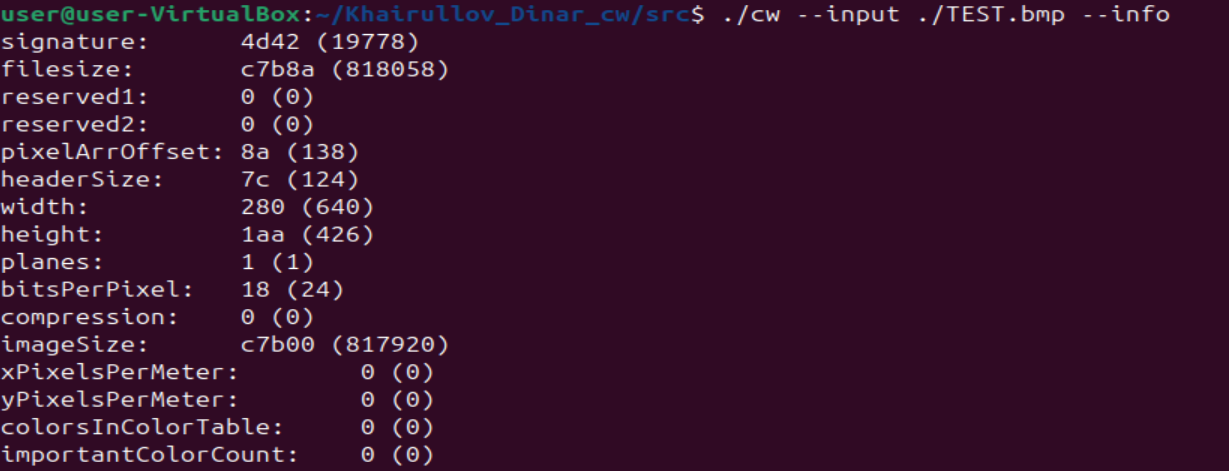


Рисунок 3 – вывод информации об изображении в консоль

Тест 3 – функция замена цвета

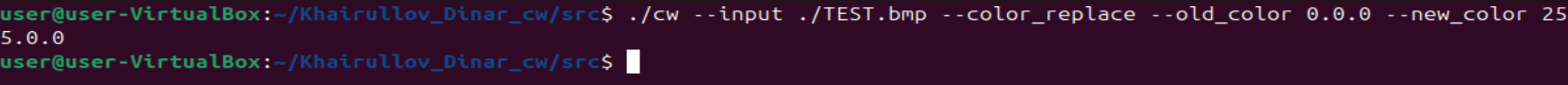


Рисунок 4 – вызов функции замены цвета в консоли



Рисунок 5 – результат работы функции замены цвета

Тест 4 – рисование прямоугольной рамки

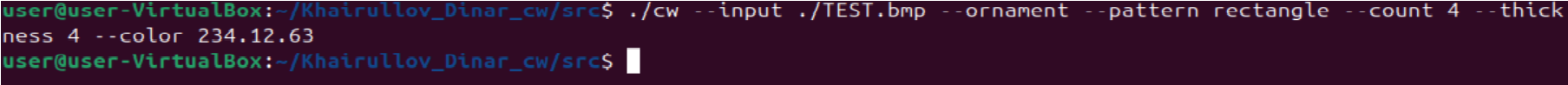


Рисунок 6 – вызов функции рисования прямоугольной рамки



Рисунок 7 – результат работы функции рисования прямоугольной рамки

Тест 5 – рисование круглой рамки

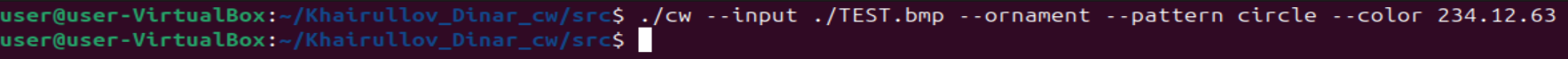


Рисунок 8 – вызов функции рисования круглой рамки



Рисунок 9 – результат работы функции рисования круглой рамки

Тест 6 – рисование рамки из полукругов

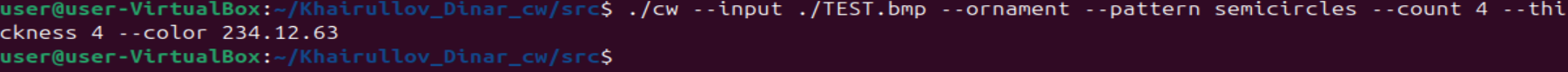


Рисунок 10 – вызов функции рисования рамки из полукругов



Рисунок 11 – результат работы функции рисования рамки из полукругов

Тест 7 – нахождения прямоугольников одного цвета

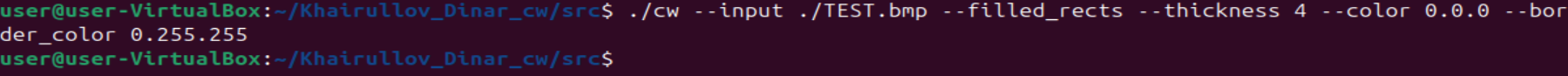


Рисунок 12 – вызов нахождения прямоугольников одного цвета



Рисунок 13 – результат работы функции нахождения прямоугольников одного цвета

**приложение б**

**Исходный код программы**

Название файла: main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <getopt.h>

#include <math.h>

#include <stdint.h>

#include <ctype.h>

#define ERROR\_OPENING\_FILE\_EXIT 40

#define ERROR\_REALOCATING\_MEMORY\_EXIT 41

#define WRONG\_BMP\_DATA\_EXIT 42

#define WRONG\_OPTION\_EXIT 43

#define ERROR\_COLOR\_EXIT 44

#define ERROR\_FEW\_ARGS\_EXIT 45

#define ERROR\_INVALID\_COUNT\_VALUE\_EXIT 46

#define ERROR\_INVALID\_THICKNESS\_VALUE\_EXIT 47

#define ERROR\_INVALID\_PATTERN\_VALUE\_EXIT 48

#define ERROR\_INVALID\_VALUE\_EXIT 49

#define ERROR\_OPENING\_FILE\_PRINT "Error: file opening error\n"

#define ERROR\_REALOCATING\_MEMORY\_PRINT "Error: realocating memory error\n"

#define WRONG\_BMP\_DATA\_PRINT "Error: wrong bmp data\n"

#define WRONG\_OPTION\_PRINT "Error: wrong option\n"

#define ERROR\_COLOR\_PRINT "Error: invalid color data\n"

#define ERROR\_FEW\_ARGS\_PRINT "Error: too few arguments were given\n"

#define ERROR\_INVALID\_COUNT\_VALUE\_PRINT "Error: invalid count value\n"

#define ERROR\_INVALID\_THICKNESS\_VALUE\_PRINT "Error: invalid thickness value\n"

#define ERROR\_INVALID\_PATTERN\_VALUE\_PRINT "Error: invalid pattern value\n"

#define ERROR\_INVALID\_VALUE\_PRINT "Error: invalid value\n"

#pragma pack(push, 1)

typedef struct

{

unsigned short signature; // определение типа файла

unsigned int filesize; // размер файла

unsigned short reserved1; // должен быть 0

unsigned short reserved2; // должен быть 0

unsigned int pixelArrOffset; // начальный адрес байта, в котором находятся данные изображения (массив пикселей)

} BitmapFileHeader;

typedef struct

{

unsigned int headerSize; // размер этого заголовка в байтах

unsigned int width; // ширина изображения в пикселях

unsigned int height; // высота изображения в пикселях

unsigned short planes; // кол-во цветовых плоскостей (должно быть 1)

unsigned short bitsPerPixel; // глубина цвета изображения

unsigned int compression; // тип сжатия; если сжатия не используется, то здесь должен быть 0

unsigned int imageSize; // размер изображения

unsigned int xPixelsPerMeter; // горизонтальное разрешение (пиксель на метр)

unsigned int yPixelsPerMeter; // вертикальное разрешение (пиксель на метр)

unsigned int colorsInColorTable; // кол-во цветов в цветовой палитре

unsigned int importantColorCount; // кол-во важных цветов (или 0, если каждый цвет важен)

} BitmapInfoHeader;

typedef struct

{

unsigned char b;

unsigned char g;

unsigned char r;

} Rgb;

#pragma pack(pop)

typedef struct

{

BitmapInfoHeader bmih;

BitmapFileHeader bmfh;

Rgb \*\*img;

} BMP;

void printFileHeader(BitmapFileHeader header);

void printInfoHeader(BitmapInfoHeader header);

void printHelp();

int getOffset(size\_t width);

void swap(int \*a, int \*b);

void color\_pixel(BMP\* bmp\_file, int x, int y, Rgb \*color);

void check\_bmp\_data(BMP\* bmp\_file);

Rgb get\_color(char\* char\_color);

BMP read\_bmp(char\* file\_name);

void write\_bmp(char file\_name[], BMP BMP\_file);

void color\_area(BMP\* bmp\_file, int x\_left, int y\_down, int x\_right, int y\_up, Rgb\* color);

void draw\_rectangle(BMP\* bmp\_file, Rgb\* color, int thickness, int x\_left, int y\_down, int x\_right, int y\_up);

void check\_color\_replace(char\* old\_color, char\* new\_color);

void color\_replace(BMP\* bmp\_file, Rgb old\_color, Rgb new\_color);

void rectangle\_ornament(BMP\* bmp\_file, Rgb\* color, int thickness, int count);

void circle\_ornament(BMP\* bmp\_file, Rgb\* color);

void printFileHeader(BitmapFileHeader header)

{

printf("signature:\t%x (%hu)\n", header.signature, header.signature);

printf("filesize:\t%x (%u)\n", header.filesize, header.filesize);

printf("reserved1:\t%x (%hu)\n", header.reserved1, header.reserved1);

printf("reserved2:\t%x (%hu)\n", header.reserved2, header.reserved2);

printf("pixelArrOffset:\t%x (%u)\n", header.pixelArrOffset, header.pixelArrOffset);

}

void printInfoHeader(BitmapInfoHeader header)

{

printf("headerSize:\t%x (%u)\n", header.headerSize, header.headerSize);

printf("width: \t%x (%u)\n", header.width, header.width);

printf("height: \t%x (%u)\n", header.height, header.height);

printf("planes: \t%x (%hu)\n", header.planes, header.planes);

printf("bitsPerPixel:\t%x (%hu)\n", header.bitsPerPixel, header.bitsPerPixel);

printf("compression:\t%x (%u)\n", header.compression, header.compression);

printf("imageSize:\t%x (%u)\n", header.imageSize, header.imageSize);

printf("xPixelsPerMeter:\t%x (%u)\n", header.xPixelsPerMeter, header.xPixelsPerMeter);

printf("yPixelsPerMeter:\t%x (%u)\n", header.yPixelsPerMeter, header.yPixelsPerMeter);

printf("colorsInColorTable:\t%x (%u)\n", header.colorsInColorTable, header.colorsInColorTable);

printf("importantColorCount:\t%x (%u)\n", header.importantColorCount, header.importantColorCount);

}

void printHelp()

{

printf("Course work for option 4.10, created by Khairullov Dinar\n\n");

printf("--help, -h: displaying help information\n");

printf("--info, -i: displaying file information\n");

printf("--input, -I: setting the input file name\n");

printf("--output, -o: setting the output file name\n");

printf("--color\_replace, -C: changing pixels' color\n");

printf("--old\_color, -O: setting old\_color value\n");

printf("--new\_color, -n: setting new color value\n");

printf("--ornament, -r: drawing ornament\n");

printf("--pattern, -p: setting pattern of ornament\n");

printf("--color, -c: setting color of filled\_rectangle\n");

printf("--thickness, -t: setting thickness of lines\n");

printf("--count, -u: setting number of ornament elements\n");

printf("--filled\_rects, -f: finding rectangles filled with given color and making borders\n");

printf("--border\_color, -b : setting border color value\n");

}

void image\_free(BMP\* bmp\_file)

{

unsigned int H = bmp\_file->bmih.height;

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

free(bmp\_file->img[i]);

}

free(bmp\_file->img);

}

int getOffset(size\_t width)

{

unsigned int offset = (width \* sizeof(Rgb)) % 4;

offset = (offset ? 4-offset : 0);

return offset;

}

void swap(int \*a, int \*b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

void color\_pixel(BMP\* bmp\_file, int x, int y, Rgb \*color)

{

bmp\_file->img[y][x].r = color->r;

bmp\_file->img[y][x].g = color->g;

bmp\_file->img[y][x].b = color->b;

}

void check\_bmp\_data(BMP\* bmp\_file)

{

if(bmp\_file->bmih.bitsPerPixel != 24 || bmp\_file->bmih.compression != 0 || bmp\_file->bmfh.signature != 0x4d42 || bmp\_file->bmih.headerSize != 40)

{

printf(WRONG\_BMP\_DATA\_PRINT);

exit(WRONG\_BMP\_DATA\_EXIT);

}

}

Rgb get\_color(char\* char\_color)

{

if(char\_color == NULL)

{

printf(ERROR\_COLOR\_PRINT);

exit(ERROR\_COLOR\_EXIT);

}

Rgb color;

int rgb\_values[3];

int counter = 0;

int check\_counter = 0;

char sep[3] =".";

if(char\_color[strlen(char\_color) - 1] == '.' || char\_color[0] == '.')

{

printf(ERROR\_COLOR\_PRINT);

exit(ERROR\_COLOR\_EXIT);

}

for(int index = 0; index < strlen(char\_color); index++)

{

if(char\_color[index] == '.')

{

if(char\_color[index + 1] == '.')

{

printf(ERROR\_COLOR\_PRINT);

exit(ERROR\_COLOR\_EXIT);

}

check\_counter++;

}

}

if(check\_counter != 2)

{

printf(ERROR\_COLOR\_PRINT);

exit(ERROR\_COLOR\_EXIT);

}

char \*pointer = strtok (char\_color, sep);

while (pointer != NULL)

{

for(int i = 0; i < strlen(pointer); i++)

{

if(!isdigit(pointer[i]))

{

printf(ERROR\_COLOR\_PRINT);

exit(ERROR\_COLOR\_EXIT);

}

}

if(atoi(pointer) < 0 || atoi(pointer) > 255)

{

printf(ERROR\_COLOR\_PRINT);

exit(ERROR\_COLOR\_EXIT);

}

rgb\_values[counter] = atoi(pointer);

counter++;

pointer = strtok (NULL,sep);

}

color.r = rgb\_values[0];

color.g = rgb\_values[1];

color.b = rgb\_values[2];

return color;

}

BMP read\_bmp(char\* file\_name){

BMP BMP\_file;

FILE \*f = fopen(file\_name, "rb");

if(f == NULL)

{

printf(ERROR\_OPENING\_FILE\_PRINT);

exit(ERROR\_OPENING\_FILE\_EXIT);

}

fread(&BMP\_file.bmfh, 1, sizeof(BitmapFileHeader), f);

fread(&BMP\_file.bmih, 1, sizeof(BitmapInfoHeader), f);

check\_bmp\_data(&BMP\_file);

fseek(f, BMP\_file.bmfh.pixelArrOffset, SEEK\_SET);

unsigned int H = BMP\_file.bmih.height;

unsigned int W = BMP\_file.bmih.width;

BMP\_file.img = (Rgb\*\*)malloc(H \* sizeof(Rgb\*));

if(BMP\_file.img == NULL)

{

printf(ERROR\_REALOCATING\_MEMORY\_PRINT);

exit(ERROR\_REALOCATING\_MEMORY\_EXIT);

}

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

{

BMP\_file.img[i] = (Rgb\*)malloc(W \* sizeof(Rgb));

if(BMP\_file.img[i] == NULL)

{

printf(ERROR\_REALOCATING\_MEMORY\_PRINT);

exit(ERROR\_REALOCATING\_MEMORY\_EXIT);

}

fread(BMP\_file.img[i], 1, W \* sizeof(Rgb) + getOffset(W), f);

}

fclose(f);

return BMP\_file;

}

void write\_bmp(char file\_name[], BMP BMP\_file){

FILE \*ff = fopen(file\_name, "wb");

if(ff == NULL)

{

printf(ERROR\_OPENING\_FILE\_PRINT);

exit(ERROR\_OPENING\_FILE\_EXIT);

}

unsigned int H = BMP\_file.bmih.height;

unsigned int W = BMP\_file.bmih.width;

fwrite(&BMP\_file.bmfh, 1, sizeof(BitmapFileHeader), ff);

fwrite(&BMP\_file.bmih, 1, sizeof(BitmapInfoHeader), ff);

fseek(ff, BMP\_file.bmfh.pixelArrOffset, SEEK\_SET);

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

fwrite(BMP\_file.img[i], 1, W \* sizeof(Rgb) + getOffset(W),ff);

}

fclose(ff);

}

void color\_area(BMP\* bmp\_file, int x\_left, int y\_down, int x\_right, int y\_up, Rgb\* color)

{

unsigned int H = bmp\_file->bmih.height;

unsigned int W = bmp\_file->bmih.width;

if (x\_left < 0 || y\_down < 0 || x\_right < 0 || y\_up < 0 || x\_left >= W || x\_right >= W || y\_down >= H || y\_up >= H)

{

return;

}

if (y\_down > y\_up)

{

swap(&y\_down, &y\_up);

}

if (x\_left > x\_right)

{

swap(&x\_left, &x\_right);

}

for(int x = x\_left; x <= x\_right; x++)

{

for(int y = y\_down; y <= y\_up; y++)

{

color\_pixel(bmp\_file, x, y, color);

}

}

}

void draw\_rectangle(BMP\* bmp\_file, Rgb\* color, int thickness, int x\_left, int y\_down, int x\_right, int y\_up)

{

color\_area(bmp\_file, x\_left, y\_down, x\_left + thickness - 1, y\_up, color);

color\_area(bmp\_file, x\_left, y\_down, x\_right, y\_down + thickness -1, color);

color\_area(bmp\_file, x\_left, y\_up - thickness + 1, x\_right, y\_up, color);

color\_area(bmp\_file, x\_right - thickness + 1, y\_down, x\_right, y\_up, color);

}

//first function

void color\_replace(BMP\* bmp\_file, Rgb old\_color, Rgb new\_color)

{

unsigned int H = bmp\_file->bmih.height;

unsigned int W = bmp\_file->bmih.width;

for(int y = 0; y < H; y++)

{

for(int x = 0; x < W; x++)

{

if(bmp\_file->img[y][x].r == old\_color.r && bmp\_file->img[y][x].g == old\_color.g && bmp\_file->img[y][x].b == old\_color.b)

{

color\_pixel(bmp\_file, x, y, &new\_color);

}

}

}

}

//second function

int ornament\_check(int count, int thickness, char\* pattern)

{

int pattern\_case = 0;

if(!strcmp(pattern, "rectangle") || !strcmp(pattern, "semicircles"))

{

if(!strcmp(pattern, "rectangle"))

{

pattern\_case = 1;

}

else if(!strcmp(pattern, "semicircles"))

{

pattern\_case = 3;

}

if(count <= 0)

{

printf(ERROR\_INVALID\_COUNT\_VALUE\_PRINT);

exit(ERROR\_INVALID\_COUNT\_VALUE\_EXIT);

}

if(thickness <= 0)

{

printf(ERROR\_INVALID\_THICKNESS\_VALUE\_PRINT);

exit(ERROR\_INVALID\_THICKNESS\_VALUE\_EXIT);

}

}

else if(!strcmp(pattern, "circle"))

{

pattern\_case = 2;

}

else

{

printf(ERROR\_INVALID\_PATTERN\_VALUE\_PRINT);

exit(ERROR\_INVALID\_PATTERN\_VALUE\_EXIT);

}

return pattern\_case;

}

void rectangle\_ornament(BMP\* bmp\_file, Rgb\* color, int thickness, int count)

{

int x\_left = 0;

int y\_down = 0;

int x\_right = bmp\_file->bmih.width - 1;

int y\_up = bmp\_file->bmih.height - 1;

while(count !=0)

{

draw\_rectangle(bmp\_file, color, thickness, x\_left, y\_down, x\_right, y\_up);

count--;

x\_left = x\_left + thickness\*2;

y\_down = y\_down + thickness\*2;

x\_right = x\_right - thickness\*2;

y\_up = y\_up - thickness\*2;

}

}

void circle\_ornament(BMP\* bmp\_file, Rgb\* color)

{

int x\_center = bmp\_file->bmih.width / 2;

int y\_center = bmp\_file->bmih.height / 2;

int radius = 0;

if(x\_center > y\_center){

radius = y\_center;

}

else{

radius = x\_center;

}

for (int x = 0; x < bmp\_file->bmih.width; x++) {

for (int y = 0; y < bmp\_file->bmih.height; y++) {

if (sqrt(pow(x - x\_center, 2) + pow(y - y\_center, 2)) > radius) {

color\_pixel(bmp\_file, x, y, color);

}

}

}

}

void semicircle\_ornament(BMP\* bmp\_file, Rgb\* color, int thickness, int count)

{

unsigned int H = bmp\_file->bmih.height;

unsigned int W = bmp\_file->bmih.width;

int horizontal\_circles\_radius = 0;

int vertical\_circles\_radius = 0;

for(int r = 0; r < W; r++)

{

if(r\*2\*(count) + (count)\*thickness > W)

{

horizontal\_circles\_radius = r;

break;

}

}

for(int r = 0; r < H; r++)

{

if(r\*2\*(count) + (count)\*thickness > H)

{

vertical\_circles\_radius = r;

break;

}

}

//horizontal\_ornament

for(int above\_down = 0; above\_down < 2; above\_down++)

{

int x\_center = horizontal\_circles\_radius + ceil(thickness/2);

int y\_center;

if(above\_down == 0)

{

y\_center = H - 1;

}

else

{

y\_center = 0;

}

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

{

for(int x = 0; x < W; x++)

{

for(int y = 0; y < H; y++)

{

double distance = sqrt(pow(abs(x - x\_center), 2) + pow(abs(y - y\_center), 2));

if (distance >= horizontal\_circles\_radius && distance <= horizontal\_circles\_radius + thickness)

{

color\_pixel(bmp\_file, x, y, color);

}

}

}

x\_center += thickness + 2\*horizontal\_circles\_radius;

}

}

//vertical\_ornament

for(int left\_right = 0; left\_right < 2; left\_right++)

{

int x\_center;

int y\_center = H - 1 - (vertical\_circles\_radius + ceil(thickness/2));

if(left\_right == 0)

{

x\_center = 0;

}

else

{

x\_center = W - 1;

}

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

{

for(int x = 0; x < W; x++)

{

for(int y = 0; y < H; y++)

{

double distance = sqrt(pow(abs(x - x\_center), 2) + pow(abs(y - y\_center), 2));

if (distance >= vertical\_circles\_radius && distance <= vertical\_circles\_radius + thickness)

{

color\_pixel(bmp\_file, x, y, color);

}

}

}

y\_center -= (thickness + 2\*vertical\_circles\_radius);

}

}

}

void colored\_pixel\_check(int\* flag, int x, int y, int colored\_count, int\*\* start\_colored\_pixels, int\*\* end\_colored\_pixels)

{

if(colored\_count == 0)

{

\*flag = 0;

return;

}

else

{

for(int t = 0; t < colored\_count; t++)

{

if(x >= start\_colored\_pixels[t][0] && x <= end\_colored\_pixels[t][0] && y >= start\_colored\_pixels[t][1] && y <= end\_colored\_pixels[t][1])

{

\*flag = 1;

break;

}

else

{

\*flag = 0;

}

}

}

}

void filled\_rects(BMP\* bmp\_file, Rgb\* color, Rgb\* border\_color, int thickness)

{

unsigned int H = bmp\_file->bmih.height;

unsigned int W = bmp\_file->bmih.width;

int x\_left = 0;

int y\_down = 0;

int x\_right = 0 ;

int y\_up = 0;

int colored\_count = 0;

int flag = 0;

int\*\* start\_colored\_pixels = malloc(sizeof(int\*)\*2);

if(start\_colored\_pixels == NULL)

{

printf(ERROR\_REALOCATING\_MEMORY\_PRINT);

exit(ERROR\_REALOCATING\_MEMORY\_EXIT);

}

start\_colored\_pixels[colored\_count] = malloc(sizeof(int) \* 3);

if(start\_colored\_pixels[colored\_count] == NULL)

{

printf(ERROR\_REALOCATING\_MEMORY\_PRINT);

exit(ERROR\_REALOCATING\_MEMORY\_EXIT);

}

int\*\* end\_colored\_pixels = malloc(sizeof(int\*)\*2);

if(end\_colored\_pixels == NULL)

{

printf(ERROR\_REALOCATING\_MEMORY\_PRINT);

exit(ERROR\_REALOCATING\_MEMORY\_EXIT);

}

end\_colored\_pixels[colored\_count] = malloc(sizeof(int)\*3);

if(end\_colored\_pixels[colored\_count] == NULL)

{

printf(ERROR\_REALOCATING\_MEMORY\_PRINT);

exit(ERROR\_REALOCATING\_MEMORY\_EXIT);

}

for(int x = 0; x < W; x++)

{

for(int y = 0; y < H; y++)

{

if(bmp\_file->img[y][x].r == color->r && bmp\_file->img[y][x].g == color->g && bmp\_file->img[y][x].b == color->b)

{

colored\_pixel\_check(&flag, x, y, colored\_count, start\_colored\_pixels, end\_colored\_pixels);

if(flag)

{

flag = 0;

break;

}

x\_left = x;

x\_right = x;

y\_down = y;

y\_up = y;

for(int t = x; t < W; t++)

{

if(bmp\_file->img[y][t].r == color->r && bmp\_file->img[y][t].g == color->g && bmp\_file->img[y][t].b == color->b)

{

colored\_pixel\_check(&flag, x, t, colored\_count, start\_colored\_pixels, end\_colored\_pixels);

if(flag)

{

flag = 0;

break;

}

x\_right = t;

}

else

{

break;

}

}

for(int z = y; z < H; z++)

{

for(int w = x\_left; w <= x\_right; w++)

{

colored\_pixel\_check(&flag, w, z, colored\_count, start\_colored\_pixels, end\_colored\_pixels);

if(flag)

{

flag = 0;

break;

}

if(!(bmp\_file->img[z][w].r == color->r && bmp\_file->img[z][w].g == color->g && bmp\_file->img[z][w].b == color->b))

{

flag = 1;

break;

}

y\_up = z;

}

if(flag)

{

flag = 0;

break;

}

}

start\_colored\_pixels[colored\_count][0] = x\_left;

start\_colored\_pixels[colored\_count][1] = y\_down;

end\_colored\_pixels[colored\_count][0] = x\_right;

end\_colored\_pixels[colored\_count][1] = y\_up;

colored\_count++;

start\_colored\_pixels = realloc(start\_colored\_pixels, sizeof(int\*)\*(colored\_count + 5));

if(start\_colored\_pixels == NULL)

{

printf(ERROR\_REALOCATING\_MEMORY\_PRINT);

exit(ERROR\_REALOCATING\_MEMORY\_EXIT);

}

start\_colored\_pixels[colored\_count] = malloc(sizeof(int)\*3);

if(start\_colored\_pixels[colored\_count] == NULL)

{

printf(ERROR\_REALOCATING\_MEMORY\_PRINT);

exit(ERROR\_REALOCATING\_MEMORY\_EXIT);

}

end\_colored\_pixels = realloc(end\_colored\_pixels, sizeof(int\*)\*(colored\_count + 5));

if(end\_colored\_pixels == NULL)

{

printf(ERROR\_REALOCATING\_MEMORY\_PRINT);

exit(ERROR\_REALOCATING\_MEMORY\_EXIT);

}

end\_colored\_pixels[colored\_count] = malloc(sizeof(int)\*3);

if(end\_colored\_pixels[colored\_count] == NULL)

{

printf(ERROR\_REALOCATING\_MEMORY\_PRINT);

exit(ERROR\_REALOCATING\_MEMORY\_EXIT);

}

//border

if((y\_down + thickness - 1 >= y\_up) || x\_left + thickness - 1 >= x\_right)

{

color\_area(bmp\_file, x\_left, y\_down, x\_right, y\_up, border\_color);

}

else

{

draw\_rectangle(bmp\_file, border\_color, thickness, x\_left, y\_down, x\_right, y\_up);

}

}

}

}

for(int counter = 0; counter <= colored\_count; counter++)

{

free(start\_colored\_pixels[counter]);

free(end\_colored\_pixels[counter]);

}

free(start\_colored\_pixels);

free(end\_colored\_pixels);

}

void gamma\_func(BMP\* bmp\_file, double value)

{

if(value <= 0)

{

printf(ERROR\_INVALID\_VALUE\_PRINT);

exit(ERROR\_INVALID\_VALUE\_EXIT);

}

unsigned int H = bmp\_file->bmih.height;

unsigned int W = bmp\_file->bmih.width;

for(int y = 0; y < H; y++)

{

for(int x = 0; x < W; x++)

{

int old\_r = bmp\_file->img[y][x].r;

int old\_g = bmp\_file->img[y][x].g;

int old\_b = bmp\_file->img[y][x].b;

double number = 255;

double new\_r = old\_r / number;

double new\_g = old\_g / number;

double new\_b = old\_b / number;

if(floor(powf( new\_r, value) \* 255) >= 0 && floor(powf(new\_r, value) \* 255) <= 255)

{

bmp\_file->img[y][x].r = (int)floor(powf( new\_r, value) \* 255);

}

if(floor(powf(new\_g, value)\*255) >= 0 && floor(powf(new\_g, value)\*255) <= 255)

{

bmp\_file->img[y][x].g = (int)floor(powf(new\_g, value)\*255);

}

if(floor(powf(new\_b, value)\*255) >= 0 && floor(powf(new\_b, value)\*255) <= 255)

{

bmp\_file->img[y][x].b = (int)floor(powf(new\_b, value)\*255);

}

}

}

}

int main(int argc, char \*argv[])

{

if(argc < 2)

{

printf(ERROR\_FEW\_ARGS\_PRINT);

exit(ERROR\_FEW\_ARGS\_EXIT);

}

const char \*short\_options = "hiI:o:CO:n:rp:c:t:u:fb:gv:";

const struct option long\_options[] =

{

{"help", no\_argument, 0, 'h'},

{"info", no\_argument, 0, 'i'},

{"input", required\_argument, 0, 'I'},

{"output", required\_argument, 0, 'o'},

{"color\_replace", no\_argument, 0, 'C'},

{"old\_color", required\_argument, 0, 'O'},

{"new\_color", required\_argument, 0, 'n'},

{"ornament", no\_argument, 0, 'r'},

{"pattern", required\_argument, 0, 'p'},

{"color", required\_argument, 0, 'c'},

{"thickness", required\_argument, 0, 't'},

{"count", required\_argument, 0, 'u'},

{"filled\_rects", no\_argument, 0, 'f'},

{"border\_color", required\_argument, 0, 'b'},

{"gamma", no\_argument, 0, 'g'},

{"value", required\_argument, 0, 'v'},

{0, 0, 0, 0}

};

char \*input\_file = argv[argc - 1];

char \*output\_file = "out.bmp";

int opt = 0;

int option\_index = 0;

int option = 0;

char\* char\_old\_color = NULL;

char\* char\_new\_color = NULL;

char\* pattern = NULL;

char\* char\_color = NULL;

int thickness = -1;

int count = -1;

char\* char\_border\_color = NULL;

double value = 0;

while ((opt = getopt\_long(argc, argv, short\_options, long\_options, &option\_index)) != -1)

{

switch (opt)

{

case 'h':

{

printHelp();

exit(0);

break;

};

case 'i':

{

option = 4;

break;

};

case 'o':

{

output\_file = optarg;

break;

};

case 'I':

{

input\_file = optarg;

break;

};

case 'C':

{

option = 1;

break;

};

case 'O':

{

char\_old\_color = optarg;

break;

};

case 'n':

{

char\_new\_color = optarg;

break;

};

case 'r':

{

option = 2;

break;

};

case 'p':

{

pattern = optarg;

break;

};

case 'c':

{

char\_color = optarg;

break;

};

case 't':

{

thickness = atoi(optarg);

break;

};

case 'u':

{

count = atoi(optarg);

break;

};

case 'f':

{

option = 3;

break;

};

case 'b':

{

char\_border\_color = optarg;

break;

};

case 'g':

{

option = 5;

break;

};

case 'v':

{

value = atof(optarg);

break;

};

case '?':

{

printf(WRONG\_OPTION\_PRINT);

exit(WRONG\_OPTION\_EXIT);

break;

}

}

}

BMP bmp\_file = read\_bmp(input\_file);

unsigned int H = bmp\_file.bmih.height;

unsigned int W = bmp\_file.bmih.width;

switch(option)

{

case 1:

{

Rgb old\_color = get\_color(char\_old\_color);

Rgb new\_color = get\_color(char\_new\_color);

color\_replace(&bmp\_file, old\_color, new\_color);

break;

}

case 2:

{

Rgb color = get\_color(char\_color);

int pattern\_case = ornament\_check(count, thickness, pattern);

switch(pattern\_case)

{

case 1:

{

rectangle\_ornament(&bmp\_file, &color, thickness, count);

break;

}

case 2:

{

circle\_ornament(&bmp\_file, &color);

break;

}

case 3:

{

semicircle\_ornament(&bmp\_file, &color, thickness, count);

break;

}

}

break;

}

case 3:

{

Rgb color = get\_color(char\_color);

Rgb border\_color = get\_color(char\_border\_color);

if(thickness <= 0)

{

printf(ERROR\_INVALID\_THICKNESS\_VALUE\_PRINT);

exit(ERROR\_INVALID\_THICKNESS\_VALUE\_EXIT);

}

filled\_rects(&bmp\_file, &color, &border\_color, thickness);

break;

}

case 4:

{

printFileHeader(bmp\_file.bmfh);

printInfoHeader(bmp\_file.bmih);

exit(0);

break;

}

case 5:

{

gamma\_func(&bmp\_file, value);

break;

}

default:

{

printf(WRONG\_OPTION\_PRINT);

exit(WRONG\_OPTION\_EXIT);

break;

}

}

write\_bmp(output\_file, bmp\_file);

image\_free(&bmp\_file);

return 0;

}