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 _____ Малиновский А.А.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Малиновский Александр

Группа: 3343

Тема: Обработка BMP изображения

Условия задания (Вариант 4.3):

Программа должна иметь следующую функции по обработке изображений:

- (1) Рисование прямоугольника. Флаг для выполнения данной операции: `--rect`. Он определяется:
  - Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
  - Координатами правого нижнего угла. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
  - 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)
  - Прямоугольник 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`)
- (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) Поворот изображения (части) на 90/180/270 градусов. Флаг для выполнения данной операции: `--rotate`. 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
    - Углом поворота. Флаг `--angle`, возможные значения: `90`, `180`, `270`

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 15.05.2024

АННОТАЦИЯ

В ходе курсовой работы реализована программа, осуществляющая обработку BMP изображения. Для взаимодействия с программой реализован интерфейс командной строки (CLI). Программа реализует следующие функции: рисование прямоугольника, рисование рамки в виде узора, поворот части изображения на 90/180/270 градусов. Сборка проекта осуществляется с помощью утилиты make.

ВВЕДЕНИЕ

Цель работы:

Разработать интерактивное консольное приложение для обработки изображений в формате BMP, которое предоставляет следующие функции:

• Считывание и запись BMP-изображений.

• Изменение изображения

• Визуализация обработанного изображения.

1. ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

Описание структур:

1. *BMPHeader* – структура, содержащая заголовок информационный заголовок BMP файла (сигнатуру, местонахождение данных растрового массива, размер файла и два зарезервированных поля.
2. *DIBHeader* – структура, содержащая данные о BMP файле(Содержит ширину, высоту и битность растра, а также формат пикселей, информацию о цветовой таблице и разрешении.
3. *Rgb* – структура, которая представляет собой цвет, кодируемый тремя компонентами: *r* (красный), *g* (зеленый), *b* (синий).
4. *BMPFile* – структура, соединяющая объекты *BMPHeader*, *DIBHeader* и *Rgb* в единую структуру файла
5. *FunctionParams* – структура, содержащая аргументы и флаги функций для обработки изображения.

Описание функций:

1. *char** parseLine(char* line,int num_val)* – разбивает строчку формата rrr.ggg.bbb или координаты (--*left_up* 123.321) на двумерный массив строк до точки или пробела.
2. *FunctionParams* initFunctionalParams(FunctionParams* fp)* – инициализирует поля объекта *FunctionParams* стандартными значениями.
3. *FunctionParams* parseCommandLine(int argc,char* argv[])* – с помощью функции *getopt_long* считывает аргументы командной строки.
4. *void checkFunctionsNumber(FunctionParams* fp)* – проверяет допустимое количество выполняемых функций (в данной программе 1).
5. *void checkRect(FunctionParams* fp,int height, int width)* – проверяет корректность аргументов для функции под флагом *–rect*.
6. *void checkOrnament(FunctionParams* fp)* – проверяет корректность аргументов для функции под флагом *–ornament*.

7. *void checkRotate(FunctionParams* fp,int height,int width)* – проверяет корректность аргументов для функции под флагом *–rotate*.
8. *int checkBMPFormat(BMPHeader bmph)*– проверяет заголовок файла на соответствие BMP формату.
9. *BMPFile* readBMP(char file_name[])*– считывает BMP файл по указанному пути.
10. *void writeBMP(char file_name[],BMPFile* bmpf)*– выполняет запись BMP файла по указанному пути.
11. *void freeBMPfile(BMPFile* bmpf)*– очищает память после работы с BMP файлом.
12. *void printFileHeader(BMPHeader header)*– выводит на экран поля объекта структуры *BMPHeader.*
13. *void printInfoHeader(DIBHeader header)*– выводит на экран поля объекта структуры *DIBHeader.*
14. *void raiseError(const char* message, int error_code)*– выводит в консоль сообщением об ошибке и выходит из программы указанным кодом ошибки.
15. *int check(int x, int y, int W, int H)*– проверяет координаты, чтобы при обработке изображения не выйти за его пределы.
16. *void drawSimpleCircle(BMPFile* bmpf,int x0, int y0, int radius, Rgb color)*– рисует в указанной точке окружность по алгоритму Брезенхэма В этом алгоритме строится дуга окружности для первого квадранта, а координаты точек окружности для остальных квадрантов получаются симметрично. На каждом шаге алгоритма рассматриваются три пикселя, и из них выбирается наиболее подходящий путём сравнения расстояний от центра до выбранного пикселя с радиусом окружности.
17. *void drawLine(BMPFile* bmpf, int x1, int y1, int x2, int y2, int line_thickness, Rgb color)* – рисует линию с заданной толщиной по алгоритму Брезенхэма используя функцию *drawSimpleCircle.*

18. *Rgb** drawRectangle(FunctionParams* fp,BMPFile* bmpf)* – рисует рамку прямоугольника из 4 линий используя функцию *drawLine,* далее заливает прямоугольник указанным цветом(если указан флаг –fill)
19. *Rgb** drawOrnament(FunctionParams* fp, BMPFile* bmpf)*– рисует орнамент на изображении в зависимости от определённого шаблона(--*pattern*) и других передаваемых параметров.
20. *Rgb** rotateImage(FunctionParams* fp,BMPFile* bmpf)* – поворачивает указанную область изображения на 90, 180 или 270 градусов против часовой стрелки.
21. *int main(char argc, char* argv[])*– главная функция программы, в ней считывается файл, который будет обрабатываться программой, параметры функций обработки, в зависимости от флага выполняется указанная функция, далее изображения записывается по указанному пути.

Архитектура разработанной программы модульная, что повышает ее масштабируемость и удобство дальнейшего развития. Функциональность программы разделена на отдельные файлы, каждый из которых отвечает за определенный аспект обработки изображения. Для сборки программы используется Makefile, упрощающий управление зависимостями между модулями и процессом компиляции. Исходный код программы представлен в приложении А.

ТЕСТИРОВАНИЕ

Рисунок 1 – изображение для тестирования

1. Тестирование функции *rect*:

   Аргументы для запуска: ./cw --fill --color 132.3.4 --output ./images/out.bmp --right_down 300.252 --rect --fill_color 248.140.177 --input ./images/Tiffany.bmp --left_up 0.52 --thickness 13

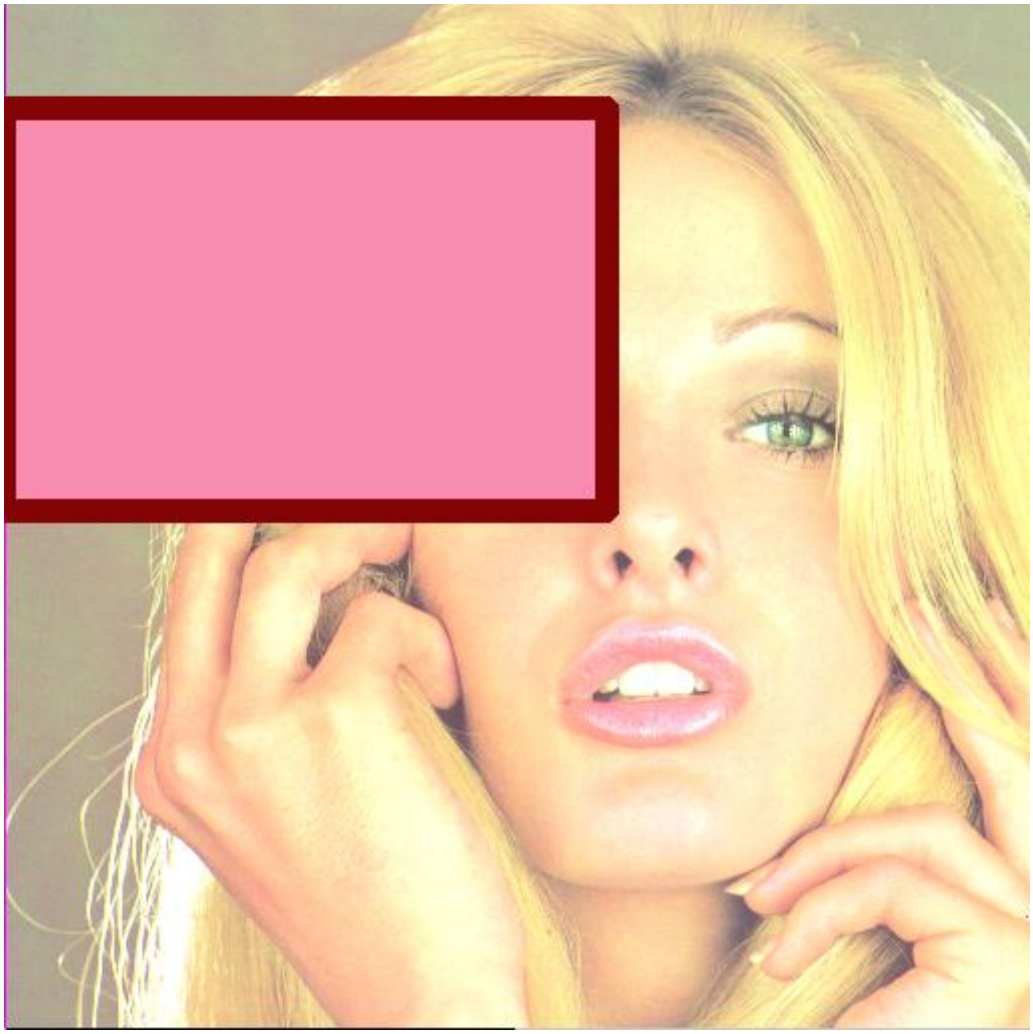

Рисунок 2 – результат работы функции *rect*

2. Тестирование функции *ornament(rect)*:

   Аргументы для запуска: ./cw --ornament --input ./images/Tiffany.bmp --output ./images/out.bmp --pattern rectangle --count 7 --thickness 15 --color 47.54.79

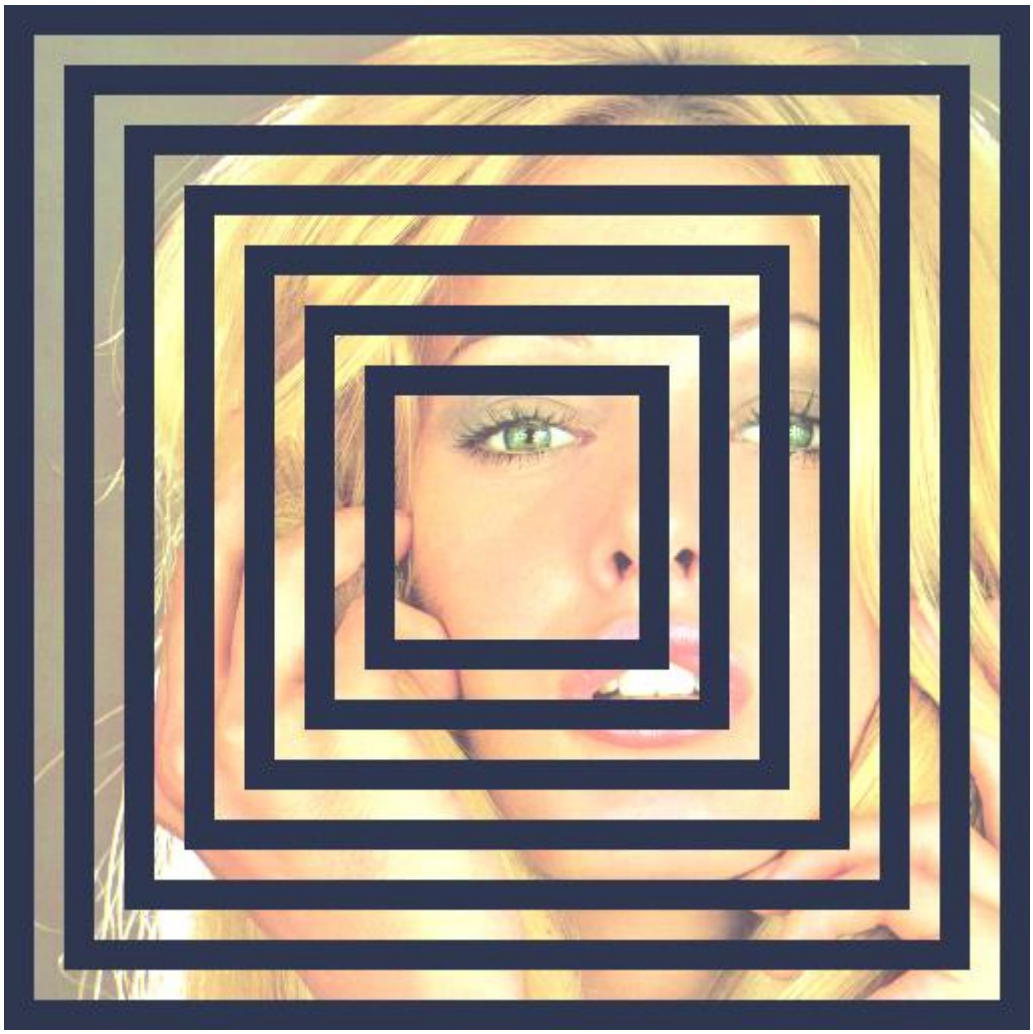

Рисунок 3 – результат работы функции *ornament (rectangle)*

3. Тестирование функции *ornament(circle)*:

   Аргументы для запуска: ./cw --ornament --input ./images/Tiffany.bmp --output ./images/out.bmp --pattern circle --count 7 --thickness 15 --color 47.54.79

Рисунок 4 – результат работы функции *ornament (circle)*

4. Тестирование функции *ornament*:

   Аргументы для запуска: : ./cw --ornament --input ./images/Tiffany.bmp --output ./images/out.bmp --pattern semicircles --count 7 --thickness 15 --color 47.54.79

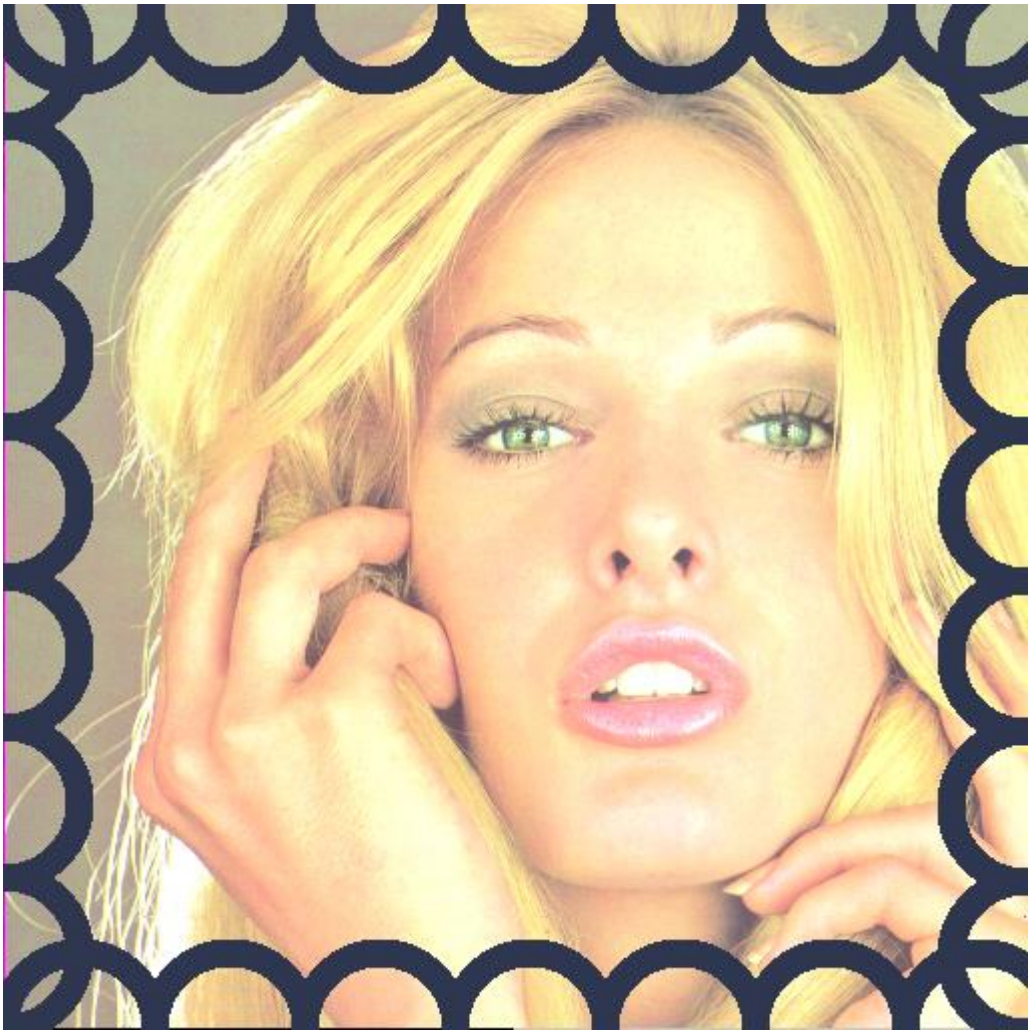

Рисунок 5 – результат работы функции *ornament (semicircles)*

5. Тестирование функции *rotate*:

Аргументы для запуска: ./cw --rotate --left_up 100.50 --right_down 400.510 --angle 90 --input ./images/Tiffany.bmp --output ./images/out.bmp

Рисунок 5 – результат работы функции *rotate (90)*

6. Тестирование функции *rotate(180)*:

Аргументы для запуска: ./cw --rotate --left_up 100.50 --right_down 400.510 --angle 180 --input ./images/Tiffany.bmp --output ./images/out.bmp

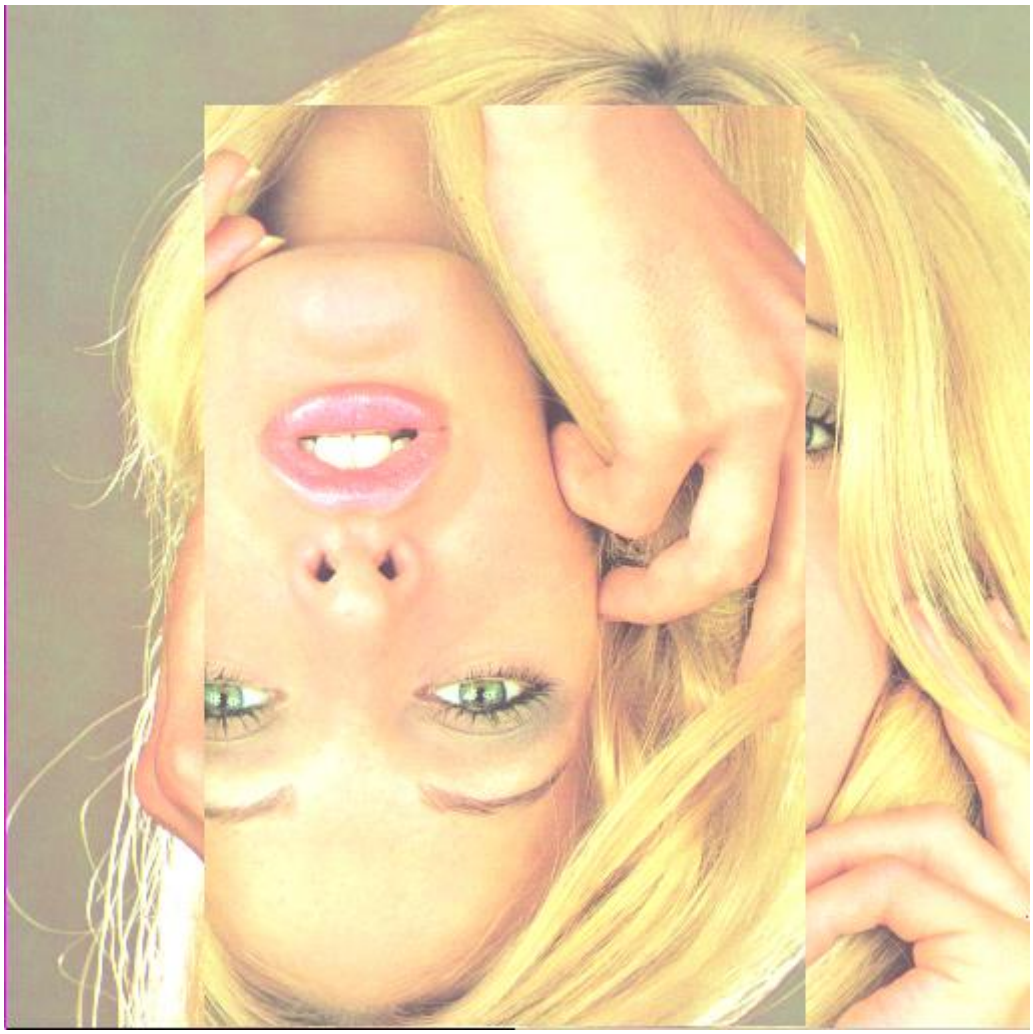

Рисунок 6 – результат работы функции *rotate (180)*

7. Тестирование функции *rotate(270)*:

Аргументы для запуска: ./cw --rotate --left_up 100.50 --right_down 400.510 --angle 270 --input ./images/Tiffany.bmp --output ./images/out.bmp

Рисунок 7 – результат работы функции *rotate (270)*

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была разработана программа на языке программирования C для обработки изображений в формате BMP. Программа предоставляет набор функций, которые могут быть выбраны пользователем через командную строку. Сборка программы выполняется с помощью утилиты make. После сборки программа запускается из командной строки, где пользователь может выбрать одну из поддерживаемых функций для обработки изображения.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: bmp.h

```
#ifndef BMP_H
#define BMP_H

#include "structures.h"

BMPFile* readBMP(char file_name[]);

int checkBMPFormat(BMPHeader bmph);

void writeBMP(char file_name[],BMPFile* bmpf);

void printFileHeader(BMPHeader header);

void printInfoHeader(DIBHeader header);

void printHelp();

void freeBMPfile(BMPFile* bmpf);

#endif
```

Название файла: errors.h

```
#ifndef ERRORS_H
#define ERRORS_H

#include <stdlib.h>
#include <stdio.h>

#define ERR_FILE_NOT_FOUND 40

#define ERR_INCORRECT_FILE_FORMAT 41

#define ERR_FILE_WRITE_ERROR 42

#define ERR_INVALID_ARGUMENT 43

#define ERR_INSUFFICIENT_ARGUMENTS 45

#define ERR_MEMORY_ALLOCATION_FAILURE 46


extern const char* color_error;

extern const char* coords_error;

extern const char* thickness_error;

extern const char* pattern_error;
```

```
extern const char* count_error;

extern const char* angle_error;


extern const char* args_error;

extern const char* multiple_func_error;

extern const char* input_file_error;

extern const char* output_file_error;

extern const char* file_type_error;

extern const char* file_opening_error;


void raiseError(const char* message, int error_code);


#endif
```

Название файла:function_params.h

```
#ifndef FUNCTION_PARAMS_H
#define FUNCTION_PARAMS_H

#include "bmp.h"

FunctionParams* parseCommandLine(int argc,char* argv[]);

FunctionParams* initFunctionalParams(FunctionParams* fp);

char** parseLine(char* line,int num_val);

void checkRect(FunctionParams* fp,int height,int width);

void checkOrnament(FunctionParams* fp);

void checkRotate(FunctionParams* fp,int height, int width);

void checkFunctionsNumber(FunctionParams* fp);

#endif
```

Название файла: processing.h

```
#ifndef PROCESSING_H
#define PROCESSING_H

#include "bmp.h"

Rgb** drawRectangle(FunctionParams* fp, BMPFile* bmpf);

Rgb** drawOrnament(FunctionParams* fp, BMPFile* bmpf);

Rgb** rotateImage(FunctionParams* fp,BMPFile* bmpf);
```

```
Rgb** rotate(FunctionParams* fp,BMPFile* bmpf);

void swapPixels(Rgb* a, Rgb* b);

#endif
```

Название файла: structures.h

```
#ifndef STRUCTURES_H
#define STRUCTURES_H

#pragma pack (push, 1)
//выравнивание структуры для bmp файла

typedef struct {
      unsigned short signature;             //Сигнатура файла BMP (2 байт)
      unsigned int filesize;                      //Размер файла (4 байт)
      unsigned short reserved1;             //Не используется (2 байт)
      unsigned short reserved2;             //Не используется (2 байт)
      unsigned int pixelArrOffset;          //Местонахождение данных
растрового массива (4 байт)
} BMPHeader;

typedef struct {
      unsigned int headerSize;              //Длина этого заголовка (4
байт)
      unsigned int width;                         //Ширина изображения (4
байт)
      unsigned int height;                  //Высота изображения (4 байт)
      unsigned short planes;                      //Число цветовых
плоскостей (2 байт)
      unsigned short bitsPerPixel;          //Бит/пиксел (2 байт)
      unsigned int compression;             //Метод сжатия (4 байт)
      unsigned int imageSize;                     //Длина растрового
массива (4 байт)
      unsigned int xPixelsPerMeter;         //Горизонтальное разрешение (4
байт)
      unsigned int yPixelsPerMeter;         //Вертикальное разрешение (4
байт)
      unsigned int colorsInColorTable;//Число цветов изображения (4 байт)
      unsigned int importantColorCount;     //Число основных цветов (4
байт)
} DIBHeader;

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

typedef struct{
    BMPHeader bmph;
    DIBHeader dibh;
    Rgb** rgb;
}BMPFile;

#pragma pop
```

```
typedef struct{
     char* input_file;
    char* output_file;
     int help;

     //rect flag
     int rect;
     int x0;
     int x1;
     int y0;
     int y1;
     int thickness;
     Rgb color;
     int fill;
     Rgb fill_color;

     //ornament flag
     int ornament;
     int pattern;
     int count;

     //rotate flag
     int rotate;
     int angle;
}FunctionParams;

#endif
```

Название файла: bmp.c

```
#include <getopt.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "bmp.h"
#include "function_params.h"
#include "errors.h"

int checkBMPFormat(BMPHeader bmph){
     if (bmph.signature!=0x4d42 )
          raiseError(file_type_error,41);
}

BMPFile* readBMP(char file_name[]) {
     FILE* f=fopen(file_name,"r");
     if (!f){
          raiseError(file_opening_error,40);
     }
     BMPFile* bmp_file=(BMPFile*)malloc(sizeof(BMPFile));
     fread(&bmp_file->bmph,sizeof(BMPHeader),1,f);
     checkBMPFormat(bmp_file->bmph);
     fread(&bmp_file->dibh,sizeof(DIBHeader),1,f);

     unsigned int H = bmp_file->dibh.height;
     unsigned int W = bmp_file->dibh.width;
     bmp_file->rgb = malloc(H * sizeof(Rgb*));
```

```
      for(int i = 0; i < H; i++){
            bmp_file->rgb[i] = malloc(W * sizeof(Rgb) + (4 - (W *
sizeof(Rgb)) % 4) % 4);
            fread(bmp_file->rgb[i], 1,W * sizeof(Rgb) + (4 - (W *
sizeof(Rgb)) % 4) % 4,f);
      }
      fclose(f);
      return bmp_file;
}

void writeBMP(char file_name[],BMPFile* bmpf) {
      FILE* f = fopen(file_name, "wb");
      fwrite(&bmpf->bmph, 1, sizeof(BMPHeader), f);
      fwrite(&bmpf->dibh, 1, sizeof(DIBHeader), f);
      int H=bmpf->dibh.height;
      int W=bmpf->dibh.width;
      for (int i = 0; i < H; i++) {
            fwrite(bmpf->rgb[i], 1, W * sizeof(Rgb) + (4 - (W *
sizeof(Rgb)) % 4) % 4,f);
      }
      fclose(f);
}

void freeBMPfile(BMPFile* bmpf){
      if(bmpf) {
            if(bmpf->rgb){
                  for (int i = 0; i < bmpf->dibh.height; i++)
                        free(bmpf->rgb[i]);
                  free(bmpf->rgb);
            }
     free(bmpf);
   };
}

void printFileHeader(BMPHeader 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(DIBHeader 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 4.3, created by Malinovskii
Aleksandr.\n"
"Usage: ./program [OPTIONS] [input_file]\n\n"
"Options:\n"
"-i, --input <filename>: Исходное изображение\n"
"-o, --outpu  <filename>: Вывод полученного изображения\n"
"-h, --help: Вывод справки\n\n"
"--rect: Рисует прямоугольник.\n"
"--left_up: Координаты левого верхнего угла прямоугольника.\n"
"--right_down: Координаты правого нижнего угла прямоугольника.\n"
"--thickness: Толщина линий прямоугольника.\n"
"--color: Цвет линий прямоугольника.\n"
"--fill: Флаг заливки прямоугольника.\n"
"--fill_color: Цвет заливки прямоугольника.\n\n"
"--ornament: Добавляет рамку в виде узора.\n"
"--pattern: Узор рамки.\n"
"--color: Цвет рамки.\n"
"--thickness: Толщина рамки.\n"
"--count: Количество элементов в рамке.\n"
"--rotate: Поворачивает изображение (или часть изображения).\n\n"
"--left_up: Координаты левого верхнего угла области поворота.\n"
"--right_down: Координаты правого нижнего угла области поворота.\n"
"--angle: Угол поворота.\n");
}
```

Название файла: errors.c

```
#include "errors.h"

const char* color_error="Error: Invalid color format";

const char* coords_error="Error: Invalid coordinates";

const char* thickness_error="Error: Invalid thickness";

const char* pattern_error="Error: Invalid pattern";

const char* count_error="Error: Invalid count";

const char* angle_error="Error: Invalid angle";


const char* args_error="Error: Invalid arguments";
```

```
const char* multiple_func_error="Error: More than one functions were
called";

const char* input_file_error="Error: No input file";

const char* output_file_error="Error: No output file";

const char* file_type_error="Error: Incorrect file type";

const char* file_opening_error="Error: Can't open file";

void raiseError(const char* message, int error_code){
    printf("%s\n",message);
    exit(error_code);
}
```

Название файла: function_params.c

```
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <getopt.h>
#include "function_params.h"
#include "structures.h"
#include "errors.h"


char** parseLine(char* line,int num_val){
    char** parsed_line=malloc(20*sizeof(char*));
    for (int i=0;i<num_val;i++){
        parsed_line[i]=malloc(strlen(line)*sizeof(char));
    }
    int current=0;
    int len=0 ;
    for (int i = 0; i < strlen(line); i++){
            if(i>0 && line[i-1]=='.'){
                parsed_line[current][len-1]='\0';
                current++;
                len=0;
            }
            parsed_line[current][len++]=line[i];
        }
    parsed_line[current][len]='\0';
    return parsed_line;
}

FunctionParams* initFunctionalParams(FunctionParams* fp){
    fp->rect=false;
    fp->fill=false;
    fp->x0=0;
     fp->x1=0;
     fp->y0=0;
     fp->y1=0;
     fp->thickness=0;
     fp->color.r=-1;fp->color.g=-1;fp->color.b=-1;
    fp->fill_color.r=-1;fp->fill_color.g=-1;fp->fill_color.b=-1;
```

```
    fp->ornament=false;
     fp->pattern=0;
     fp->count=0;

     fp->rotate=false;
     fp->angle=0;

    fp->help=true;
    fp->input_file=NULL;
    fp->output_file=NULL;
}

FunctionParams* parseCommandLine(int argc,char* argv[]){

    opterr=0;
    const char* short_options = "hi:o:";
    const struct option long_options[] = {
            {"help", 0, NULL, 'h'},
            {"input", 1, NULL, 'i'},
            {"output", 1, NULL, 'o'},
            {"rect", 0, NULL, 256},
            {"left_up", 1, NULL, 257},
            {"right_down", 1, NULL, 258},
            {"thickness", 1, NULL, 259},
            {"color", 1, NULL, 260},
            {"fill", 0, NULL, 261},
            {"fill_color", 1, NULL, 262},
            {"ornament", 0, NULL, 263},
            {"pattern", 1, NULL, 264},
            {"count", 1, NULL, 265},
            {"rotate", 0, NULL, 266},
            {"angle", 1, NULL, 267},
            {NULL, 0, NULL, 0}
     };

    FunctionParams* fp=malloc(sizeof(FunctionParams));
    initFunctionalParams(fp);

    int result=0;
    while ((result = getopt_long(argc, argv, short_options, long_options,
NULL)) != -1){
        switch (result)
        {
        case 'h'://-help
            printHelp();
            break;

        case 'i'://-input
            fp->input_file=optarg;
            break;

        case 'o'://-output
            fp->output_file=optarg;
            break;

        case 256://--rect
            fp->rect=true;
            break;
```

```
case 257://--left_up
    fp->x0 = strtol(parseLine(optarg,2)[0], NULL, 10);
    fp->y0= strtol(parseLine(optarg,2)[1], NULL, 10);
    break;

case 258://--right_down
    fp->x1 = strtol(parseLine(optarg,2)[0], NULL, 10);
    fp->y1= strtol(parseLine(optarg,2)[1], NULL, 10);
    break;

case 259://--thickness
    fp->thickness=strtol(optarg,NULL,10);
    break;

case 260://--color
    fp->color.r=strtol(parseLine(optarg,3)[0], NULL, 10);
    fp->color.g=strtol(parseLine(optarg,3)[1], NULL, 10);
    fp->color.b=strtol(parseLine(optarg,3)[2], NULL, 10);
    break;

case 261://--fill
    fp->fill=true;
    break;

case 262://--fill_color
    fp->fill_color.r=strtol(parseLine(optarg,3)[0], NULL, 10);
    fp->fill_color.g=strtol(parseLine(optarg,3)[1], NULL, 10);
    fp->fill_color.b=strtol(parseLine(optarg,3)[2], NULL, 10);
    break;

case 263://--ornament
    fp->ornament=true;
    break;

case 264://--pattern
    if(strcmp(optarg,"rectangle")==0){
        fp->pattern=1;
    }
    else if(strcmp(optarg,"circle")==0){
        fp->pattern=2;
    }
    else if(strcmp(optarg,"semicircles")==0){
        fp->pattern=3;
    }
    break;

case 265://--count
    fp->count=strtol(optarg,NULL,10);;
    break;

case 266://--rotate
    fp->rotate=true;
    break;

case 267://--angle
    fp->angle=strtol(optarg,NULL,10);
    break;
```

```
        case '?':
            raiseError(args_error,43);
            break;

        default:

            break;
        }
     }

    checkFunctionsNumber(fp);
    if(!fp->input_file)
        exit(0);
    if(!fp->output_file)
        exit(0);
    return fp;
}

void checkFunctionsNumber(FunctionParams* fp){
    int count=0;
    if(fp->rect)
        count++;
    if(fp->ornament)
        count++;
    if(fp->rotate)
        count++;
    if (count>1)
        raiseError(multiple_func_error,43);
}

void checkRect(FunctionParams* fp,int height, int width){
    if(fp->thickness<=0)
        raiseError(thickness_error,43);
    if (fp->color.r>255 || fp->color.g>255 ||fp->color.b>255 ||
    fp->color.r<0 || fp->color.g<0 ||fp->color.b<0)
        raiseError(color_error,43);
    if(fp->fill){
        if (fp->fill_color.r>255 || fp->fill_color.g>255
||fp->fill_color.b>255||
        fp->fill_color.r<0 || fp->fill_color.g<0 ||fp->fill_color.b<0)
            raiseError(color_error,43);
    }
}

void checkOrnament(FunctionParams* fp){
    if (fp->pattern==0)
        raiseError(pattern_error,43);
    if(fp->thickness<=0)
        raiseError(thickness_error,43);
    if(fp->count<=0)
        raiseError(count_error,43);
    if (fp->color.r>255 || fp->color.g>255 || fp->color.b>255||
    fp->color.r<0 || fp->color.g<0 || fp->color.b<0)
        raiseError(color_error,43);
}

void checkRotate(FunctionParams* fp,int height,int width){
```

```
    if(fp->x0<0)
        fp->x0=0;

    if(fp->x1<0)
        fp->x1=0;

    if(fp->y0<0)
        fp->y0=0;

    if(fp->y1<0)
        fp->y1=0;

     if(fp->x0>=width)
        fp->x0=width-1;

    if(fp->x1>=width)
        fp->x1=width-1;

    if(fp->y0>=height)
        fp->y0=height-1;

    if(fp->y1>=height)
        fp->y1=height-1;

    if (fp->angle!=90 && fp->angle!=180 && fp->angle!=270)
       raiseError(angle_error,43);
}
```

Название файла: main.c

```
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "bmp.h"
#include "processing.h"
#include "function_params.h"

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

   FunctionParams* fp=parseCommandLine(argc,argv);
   BMPFile* bmpf=readBMP(fp->input_file);

   if (fp->rect){
      checkRect(fp,bmpf->dibh.height,bmpf->dibh.width);
      bmpf->rgb=drawRectangle(fp,bmpf);
   }
   if (fp->ornament){
      checkOrnament(fp);
      bmpf->rgb=drawOrnament(fp,bmpf);
  }


   if (fp->rotate){
      checkRotate(fp,bmpf->dibh.height,bmpf->dibh.width);
      bmpf->rgb=rotateImage(fp,bmpf);
   }
```

```
   writeBMP(fp->output_file,bmpf);

   freeBMPfile(bmpf);
   free(fp);
   return 0;
}
```

Название файла: processing.c

```
#include <stdio.h>
#include<string.h>
#include <stdlib.h>
#include <math.h>
#include <errors.h>
#include "processing.h"
#include "bmp.h"

#define min(a, b) (((a) < (b)) ? (a) : (b))
#define max(a, b) (((a) > (b)) ? (a) : (b))

int check(int x, int y, int W, int H)
{
    return x >= 0 && x < W && y >= 0 && y < H;
}

void drawSimpleCircle(BMPFile* bmpf,int x0, int y0, int radius, Rgb
color){
    int D = 3 - 2 * radius;
    int x = 0;
    int y = radius;
     int W=bmpf->dibh.width;
     int H=bmpf->dibh.height;
    while (x <= y) {
           if (check(x+x0,y+y0,W,H))
           bmpf->rgb[y+y0][x+x0]=color;

           if (check(y+x0,x+y0,W,H))
                 bmpf->rgb[x+y0][y+x0]=color;

           if (check(-y+x0,x+y0,W,H))
                 bmpf->rgb[x+y0][-y+x0 ]=color;

           if (check(y+y0,-x+x0,W,H))
           bmpf->rgb[y+y0][-x+x0]=color;

           if (check(-y+y0,-x+x0,W,H))
           bmpf->rgb[-y+y0][-x+x0]=color;

           if (check(-y+x0,-x+y0,W,H))
                 bmpf->rgb[-x+y0][-y+x0]=color;

           if (check(y+x0,-x+y0,W,H))
           bmpf->rgb[-x+y0][y+x0]=color;

           if (check(x+x0,-y+y0,W,H))
           bmpf->rgb[-y+y0][x+x0]=color;
```

```
            if (D < 0) {
              D += 4 * x + 6;
              x++;
          } else {
              D += 4 * (x - y) + 10;
              x++;
              y--;
          }
      }
}

void drawLine(BMPFile* bmpf, int x1, int y1, int x2, int y2, int
line_thickness, Rgb color){
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    int sx = x1 < x2 ? 1 : -1;
    int sy = y1 < y2 ? 1 : -1;
    int err = dx - dy;
    int h = bmpf->dibh.height;
    int w = bmpf->dibh.width;
    while(1){
        if (y1 >= 0 && y1 <= h && x1 >= 0 && x1 <= w){
            if (line_thickness == 1){
                    bmpf->rgb[y1][x1]=color;
            }
        }

        if(line_thickness > 1 && x1 - (line_thickness/2) < w && y1 -
(line_thickness/2) < h && x1 + (line_thickness/2) >= 0 && y1 +
(line_thickness/2) >= 0){
            drawSimpleCircle(bmpf, x1, y1, line_thickness/2 ,color);
        }

        if (x1 == x2 && y1 == y2){
            break;
        }

        int e2 = 2 * err;
        if (e2 > -dy) {
            err -= dy;
            x1 += sx;
        }

        if (e2 < dx) {
            err += dx;
            y1 += sy;
        }
    }
}


Rgb** drawRectangle(FunctionParams* fp,BMPFile* bmpf){
     int x0=min(fp->x0,fp->x1);
     int x1=max(fp->x0,fp->x1);
     int y0=min(fp->y0,fp->y1);
     int y1=max(fp->y0,fp->y1);
```

```
      y0 = bmpf->dibh.height-y0-1;
      y1 = bmpf->dibh.height-y1-1;

      drawLine(bmpf, x0, y0, x1, y0, fp->thickness, fp->color);
     drawLine(bmpf, x0, y1, x1, y1, fp->thickness, fp->color);
     drawLine(bmpf, x1, y1, x1, y0, fp->thickness, fp->color);
     drawLine(bmpf, x0, y1, x0, y0, fp->thickness, fp->color);

      if (fp->fill){

                  for (int i = y1+fp->thickness/2; i <= y0-
fp->thickness/2; i++) {
                        for (int j = x0+fp->thickness/2; j <= x1-
fp->thickness/2; j++){
                              if
(check(j,i,bmpf->dibh.width,bmpf->dibh.height))
                                    bmpf->rgb[i][j]=fp->fill_color;
                              }
                  }
            }
      return bmpf->rgb;
}

Rgb** drawOrnament(FunctionParams* fp, BMPFile* bmpf){

      switch (fp->pattern)
      {
      case 1://pattern rectangle
      {
            int down_border=0;
            int left_border=0;
            int up_border=bmpf->dibh.height-1;
            int right_border=bmpf->dibh.width-1;

            for (int l = 0; l < fp->count; l++){
                  for (int k=0;k<fp->thickness;k++){
                        for (int i = down_border; i <= up_border; i++) {
                              for (int j = left_border; j <=right_border ;
j++){
                                    if(i==down_border|| i==up_border ||
j==left_border||j==right_border){
                                          bmpf->rgb[i][j]=fp->color;
                                    }
                              }
                        }
                        down_border+=1;
                        up_border-=1;
                        left_border+=1;
                        right_border-=1;
                  }
            down_border+=fp->thickness;
            up_border-=fp->thickness;
            left_border+=fp->thickness;
            right_border-=fp->thickness;
            }
      }
            break;
      case 2://pattern circle
```

```
     {
     int centerX = bmpf->dibh.width / 2;
    int centerY = bmpf->dibh.height / 2;
    int radius = (centerX <centerY) ? centerX : centerY;

     for (int i = 0; i < bmpf->dibh.height; i++) {
            for (int j = 0; j < bmpf->dibh.width; j++) {
                double distance = sqrt(pow((j - centerX),2) + pow((i -
centerY),2));
                if (distance > radius)
                          bmpf->rgb[i][j]=fp->color;
            }
     }
          break;
     }
     case 3://pattern semicircle
     {
          double width = (double)(bmpf->dibh.width - fp->count *
fp->thickness) / (fp->count * 2);
        double height = (double)(bmpf->dibh.height - fp->count *
fp->thickness) / (fp->count * 2);
        int radiusX = ceil(width);
        int radiusY = ceil(height);

        int count = 0;
        int middleX[fp->count * 4];
        int middleY[fp->count * 4];

        //Верхние координаты
        int current =  fp->thickness / 2 + radiusX + 1;
        for (int i = 0; i < fp->count; ++i) {
            middleX[count] = current;
            middleY[count++] = 0;
            current += radiusX + fp->thickness + radiusX;
        }

        //Левые координаты
        current = fp->thickness / 2 + radiusY + 1;
        for (int i = 0; i < fp->count; ++i) {
            middleX[count] = 0;
            middleY[count++] = current;
            current += radiusY + fp->thickness + radiusY;
        }

        //Правые координаты
        current = fp->thickness / 2 + radiusY - 1;
        for (int i = 0; i < fp->count; ++i) {
            middleX[count] = bmpf->dibh.width - 1;
            middleY[count++] = current;
            current += radiusY + fp->thickness + radiusY;
        }

        //Нижние координаты
        current = fp->thickness / 2 + radiusX - 1;
        for (int i = 0; i < fp->count; ++i) {
            middleX[count] = current;
            middleY[count++] = bmpf->dibh.height - 1;
            current += radiusX + fp->thickness + radiusX;
```

```
        }

        for (int y = 0; y < bmpf->dibh.height; ++y) {

            for (int x = 0 ; x < bmpf->dibh.width; ++x) {

                for (int i = 0; i < fp->count*4; ++i) {
                    int length = sqrt(pow(x - middleX[i], 2) + pow(y -
middleY[i], 2));
                    if ((middleX[i] == 0 || middleX[i] ==
bmpf->dibh.width - 1) && length >= radiusY && length <= radiusY +
fp->thickness) {
                                bmpf->rgb[y][x]=fp->color;
                    }

                    if ((middleY[i] == 0 || middleY[i] ==
bmpf->dibh.height - 1) && length >= radiusX && length <= radiusX +
fp->thickness) {
                        bmpf->rgb[y][x]=fp->color;
                    }
                }
            }
        }
           break;
     }

     default:
          break;
     }

     return bmpf->rgb;
}

void swapPixels(Rgb* a, Rgb* b){
     Rgb temp;
     temp=*a;
     *a=*b;
     *b=temp;
}

Rgb** rotateImage(FunctionParams* fp,BMPFile* bmpf){

          int H=bmpf->dibh.height;
          int W=bmpf->dibh.width;

          int area_H = max(fp->y0,fp->y1) - min(fp->y0,fp->y1);
     int area_W = max(fp->x0,fp->x1) - min(fp->x0,fp->x1);

          int centerX = (fp->x0 +fp->x1) / 2;
          int centerY = (fp->y0+fp->y1) / 2;

          int start_x = centerX - area_H / 2;
     int start_y = H - (centerY - area_W / 2) - 1;


          fp->y0 = H-fp->y0-1;
          fp->y1 = H-fp->y1-1;
```

```
            int y0=max(fp->y0,fp->y1);
            int y1=min(fp->y0,fp->y1);
            int x0=min(fp->x0,fp->x1);
            int x1=max(fp->x0,fp->x1);

            Rgb black={0,0,0};
            Rgb **area = malloc(sizeof( Rgb *) * (area_H ));
            for (int i = 0; i <  area_H ; i++)
            {
                  area[i] = malloc(sizeof( Rgb) * (area_W));
                  for (int j = 0; j < area_W; j++)
                  {
                        area[i][j] = bmpf->rgb[i + y1+1][j + x0];
                  }
            }
            switch (fp->angle)
            {
            case 90:
                  for (int y = 0; y < area_W; y ++) {
                  for (int x = 0; x < area_H; x ++) {
                              if(check( start_x + (area_H - x - 1),start_y -
(area_W - y - 1),W,H))
                                    bmpf->rgb[start_y - (area_W - y -
1)][ start_x + (area_H - x - 1)]=area[x][y];
                  }
              }
                  break;
            case 180:
                  for (int y = y0; y > y1; y --) {
                  for (int x = x0; x < x1; x ++) {
                              bmpf->rgb[y][x]=area[(y0 - y)][area_W - (x -
x0) - 1];
              }
          }
                  break;
            case 270:
                    for (int y = 0; y < area_W; y ++) {
                      for (int x = 0; x < area_H; x ++) {
                                  if (check(start_x + x,start_y - y,W,H))
                                        bmpf->rgb[start_y - y][start_x +
x]=area[x][y];
                  }
                    }
                  break;
            default:
                  break;
            }

            /*for (int i = RightY; i < LeftY; i++)
            {
                  for (int j = LeftX; j < RightX; j++)
                  {
                        int X = round((j - centerX) * cos(ang) - (i -
centerY) * sin(ang) + centerX),
                              Y = round((j - centerX) * sin(ang) + (i -
centerY) * cos(ang) + centerY);

                        if(Y<H && Y>=0 && X>=0 && X<W)
```

```
                              bmpf->rgb[Y][X]=area[i - RightY][j - LeftX];
                    }
               }
     */

     for (int i = 0; i < (y0 - y1); i++)
     {
          free(area[i]);
     }
     free(area);

     return bmpf->rgb;
}
```

Название файла: Makefile

```
CC = gcc
RM = rm -rf

CFLAGS = -I$(INCDIR)
LIB =-lm

SRCDIR = src
INCDIR = include
OBJDIR = obj

SOURCES = $(wildcard $(SRCDIR)/*.c)
OBJECTS = $(patsubst $(SRCDIR)/%.c, $(OBJDIR)/%.o, $(SOURCES))
EXECUTABLE = cw

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
     $(CC) $(CFLAGS) $(OBJECTS) -o $@ $(LIB)

$(OBJDIR)/%.o: $(SRCDIR)/%.c
     @mkdir -p $(OBJDIR)
     $(CC) $(CFLAGS) -c $< -o $@

clean:
     $(RM) -r $(OBJDIR) $(EXECUTABLE)
```