

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

**КУРСОВАЯ РАБОТА**  
по дисциплине «Программирование»  
**Тема: Обработка bmp-файлов**

Студент гр. 9303

---

Куршев Е.О.

Преподаватель

---

Чайка К.В.

Санкт-Петербург

2019

**ЗАДАНИЕ**  
**НА КУРСОВУЮ РАБОТУ**

Студент Куршев Е. О.

Группа 9303

Тема работы : Обработка bmp-файлов

Исходные данные:

Программе на вход подается bmp-файл.

Содержание пояснительной записи:

«Аннотация», «Содержание», «Введение», «Описание кода программы»,  
«Примеры работы программы», «Заключение», «Список использованных  
источников».

Предполагаемый объем пояснительной записи:

Не менее 50 страниц.

Дата выдачи задания: 01.03.2020

Дата сдачи реферата: 28.05.2020

Дата защиты реферата: 28.05.2020

Студент

---

Куршев Е.О.

Преподаватель

---

Чайка К.В.

## АННОТАЦИЯ

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

1. Рисование квадрата с диагоналями. Квадрат определяется:

Координатами левого верхнего угла

Размером стороны

Толщиной линий

Цветом линий

Может быть залит или нет (диагонали располагаются “поверх” заливки)

Цветом которым он залит, если пользователем выбран залитый

2. Фильтр rgb-компонент. Этот инструмент должен позволять для всего изображения либо установить в 0 либо установить в 255 значение заданной компоненты. Функционал определяется

Какую компоненту требуется изменить

В какой значение ее требуется изменить

3. Поворот изображения (части) на 90/180/270 градусов. Функционал определяется

Координатами левого верхнего угла области

Координатами правого нижнего угла области

Углом поворота

4. Рисование окружности. Окружность определяется:

либо координатами левого верхнего и правого нижнего угла квадрата, в который она вписана, либо координатами ее центра и радиусом

толщиной линии окружности

цветом линии окружности

окружность может быть залитой или нет

цветом которым залита сама окружность, если пользователем выбрана залитая окружность

# **СОДЕРЖАНИЕ**

## **Введение**

### **1 Описание функций**

- 1.1 main.
- 1.2 Вызов помощи
- 1.3 Рисование квадрата
- 1.4 Фильтр
- 1.5 Поворот изображения
- 1.6 Рисование окружности
- 1.7 Очистка памяти
- 1.8 Информация о файле
- 1.9 Чтение изображения
- 1.10 Замена заголовка BITMAPINFOV3
- 1.11 Запись изображения

### **2 Описание дополнительных компонентов**

- 2.1 Структуры
- 2.2 Библиотеки

### **3 Тестирование**

### **4 Заключение**

### **5 Использованные источники**

### **Приложение А. Код программы.**

## **ВВЕДЕНИЕ**

### **Цель работы**

Создание программы для bmp-файлов.

Для достижения поставленной цели требуется решить следующие задачи:

1) Изучить функции для работы с bmp-файлами.

2) Поэтапное выполнение мини-подзадач, которые будут решать ту или определенную мини-задачу. Например, считывание изображения, работа с пиксельными данными.

3) Группировка мини-подзадач в одну для решение определенной задачи.

4) Написание Make-файла.

5) Тестирование программы.

### **Основные теоретические положения**

*Функции для динамического выделения памяти:*

`void* calloc (size_t num, size_t size)` — функция для чистого выделения памяти.

`void* malloc (size_t size)` — функция для выделения памяти.

`void* realloc (void* ptr, size_t size)` — функция для перераспределения выделенной памяти.

`void free (void* ptr)` — функция для освобождения выделенной памяти.

*Управляющие конструкции:*

Оператор IF — ELSE используется при необходимости сделать выбор.

Формально синтаксис имеет вид

IF (выражение)

    оператор-1

ELSE

    оператор-2,

Где часть ELSE является необязательной. Сначала вычисляется выражение; если оно "истинно", то выполняется оператор1. Если оно ложно, и если есть часть с ELSE, то вместо оператора-1 выполняется оператор-2.

*Циклы:*

В конструкции

## WHILE (выражение)

оператор

вычисляется выражение. Если его значение отлично от нуля, то выполняется оператор и выражение вычисляется снова. Этот цикл продолжается до тех пор, пока значение выражения не станет нулем, после чего выполнение программы продолжается с места после оператора.

Оператор

FOR (выражение 1; выражение 2; выражение 3) оператор эквивалентен последовательности

выражение 1;

WHILE (выражение 2) оператор выражение 3;

*Оператор выбора switch.*

```
switch (выражение) {  
    case константа_1 : операторы ; break;  
    ...  
    case константа_n : операторы ; break;
```

```
    default : операторы ; break; }
```

Оператор switch вычисляет выражение и переходит к первому совпадающему значению после case. Далее выполняются операторы этого блока и по команде break происходит выход из структуры. Если ни одно из значений не совпадает с константами из case, то выполняются операторы блока default.

Оператор break - это выход из цикла или конструкции switch.

Оператор continue - переход на конец цикла (т.е. пропуск всех операторов от continue до конца структуры цикла).

# 1. Описание функций

## 1.1 main.

Точка входа в программу. В ней происходит считывание аргументов, а также выполняются выбранные пользователем функции.

В функции main реализована функция getopt\_long, благодаря которой в программу передаются аргументы командной строки (реализация CLI).

## 1.2 Вызов помощи

- void help();

Функция помогает соориентироваться пользователю в программе, печатает все возможные ключи, для того, чтобы пользователь мог правильно вызвать программу.

## 1.3 Рисование квадрата

- void MakeSquareWithoutFill(BITMAPINFOV3\* bmiV3, RGB\*\* image, int x, int y, int l, int d, int r, int g, int b);

Функция, в которой происходит рисование квадрата без заливки, по заданным координате, длине стороны, ширине и цвету.

- void MakeSquareWithFill(BITMAPINFOV3\* bmiV3, RGB\*\* image, int x, int y, int l, int w, int r, int g, int b, int r\_z, int g\_z, int b\_z);

Функция, в которой происходит рисование квадрата с заливкой, по заданным координате, длине стороны, ширине и цвету, а также по цвету заливки.

## 1.4 Фильтр

- void component(BITMAPINFOV3\* bmiV3, RGB\*\* image, char\* s, int component\_value);

Функция - фильтр. Меняет значение выбранной пользователем компоненты на введённое им значение.

## 1.5 Поворот изображения

- int TurnImage(BITMAPFILEHEADER\* bmfh, BITMAPINFOV3\* bmiV3, RGB\*\* image, int x1, int y1, int x2, int y2, int alpha, char\* outputfile);

Функция, которая поворачивает изображение (часть) на введённый пользователем угол. В функции может происходить запись в файл, если поворачивается всё изображение сразу. Возвращает значение 1, если был записан файл, иначе 0.

## 1.6 Рисование окружности

- void CircleFromCenterWithoutFill(BITMAPINFOV3\* bmiV3, RGB\*\* image, int x\_c, int y\_c, int radius, int r, int g, int b, int w);

Функция рисования окружности по координатам центра и радиусу без заливки.

- void CircleFromCenterWithFill(BITMAPINFOV3\* bmiV3, RGB\*\* image, int x\_c, int y\_c, int radius, int r, int g, int b, int w, int r\_z, int g\_z, int b\_z);

Функция рисования окружности по координатам центра с заливкой.

- void CircleFromSquareWithoutFill(BITMAPINFOV3\* bmiV3, RGB\*\* image, int x1, int y1, int x2, int y2, int r, int g, int b, int d);

Функция рисования окружности вписанной в квадрат без заливки.

Квадрат задаётся координатами углов.

- void CircleFromSquareWithFill(BITMAPINFOV3\* bmiV3, RGB\*\* image, int x1, int y1, int x2, int y2, int r, int g, int b, int d, int r\_z, int g\_z, int b\_z);

Функция рисования окружности вписанной в квадрат с заливкой. Квадрат задаётся координатами углов.

## 1.7 Очистка памяти

- void FreeMemory(RGB\*\* image, BITMAPINFOV3\* bmiV3);

Функция для очистки динамически выделенной памяти.

## 1.8 Информация о файле.

- void printFileHeader(BITMAPFILEHEADER header);

Функция, печатающая информацию о BITMAPFILEHEADER.

BITMAPFILEHEADER — 14-байтная структура, которая располагается в самом начале файла.

- void printInfoHeader(BITMAPINFOV3 header);

Функция, печатающая информацию о BITMAPINFOV3 (3 версия формата bmp). BITMAPINFOV3 - блок, который содержит:

Информационные поля.

Битовые маски для извлечения значений цветовых каналов (опциональные).

Таблица цветов (опциональная).

## 1.9 Чтение изображения

- RGB\*\* ReadImage(BITMAPINFOV3\* bmiV3, FILE\* file);

Функция чтения изображения, а именно пиксельных данных.

## 1.10 Замена заголовка BITMAPINFOV3

- BITMAPINFOV3\* swap(BITMAPINFO\* bmi, FILE\* file);

Функция, которая записывает данные в структуру BITMAPINFOV3, если входное изображение имеет верный формат.

## 1.11 Запись изображения

- void WriteImage(BITMAPFILEHEADER\* bmfh, BITMAPINFOV3\* bmiV3, RGB\*\* image, char\* outputfile);

Функция для записи изображения в файл, введённый пользователем.

# 2. Описание дополнительных компонентов

## 2.1 Структуры

- Структура RGB

Хранит значения компонент пикселя.

- Структура BITMAPINFOV3

Хранит информацию о файле.

- Структура BITMAPFILEHEADER.

Хранит информацию о заголовке bmp-файла.

- Структура BITMAPINFO

Универсальная структура, которая характерна для всех файлов. По ней происходит проверка версии bmp-файла.

- Структура BITMAPINFOV3\_tmp

Временная структура. Нужна для записи файла, если он имеет верный формат.

- Структура Commands

Структура для получения аргументов из функции getopt\_long.

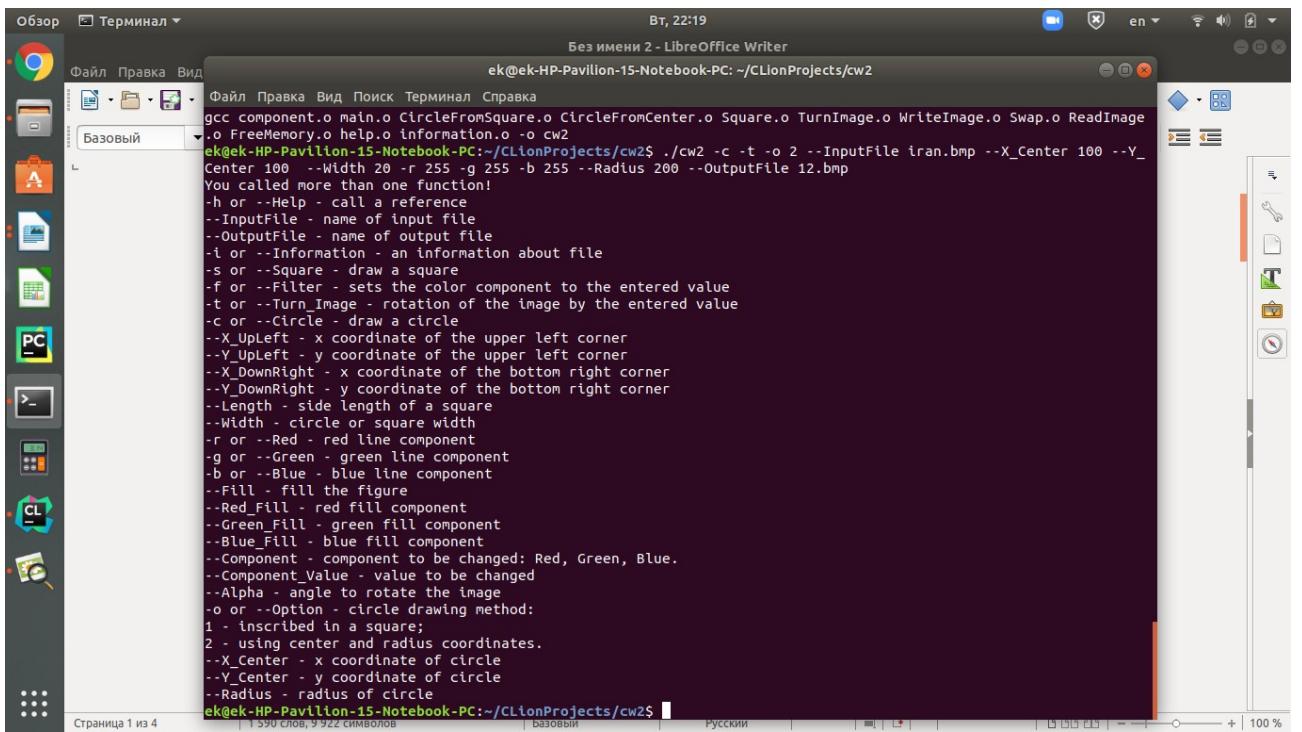
## 2.2 Библиотеки

- libraries.h

Файл, со всеми используемыми библиотеками языка C.

## 3. Тестирование

Несколько функций одновременно



The screenshot shows a terminal window within the LibreOffice suite. The terminal title is "Без имени 2 - LibreOffice Writer". The command entered is:

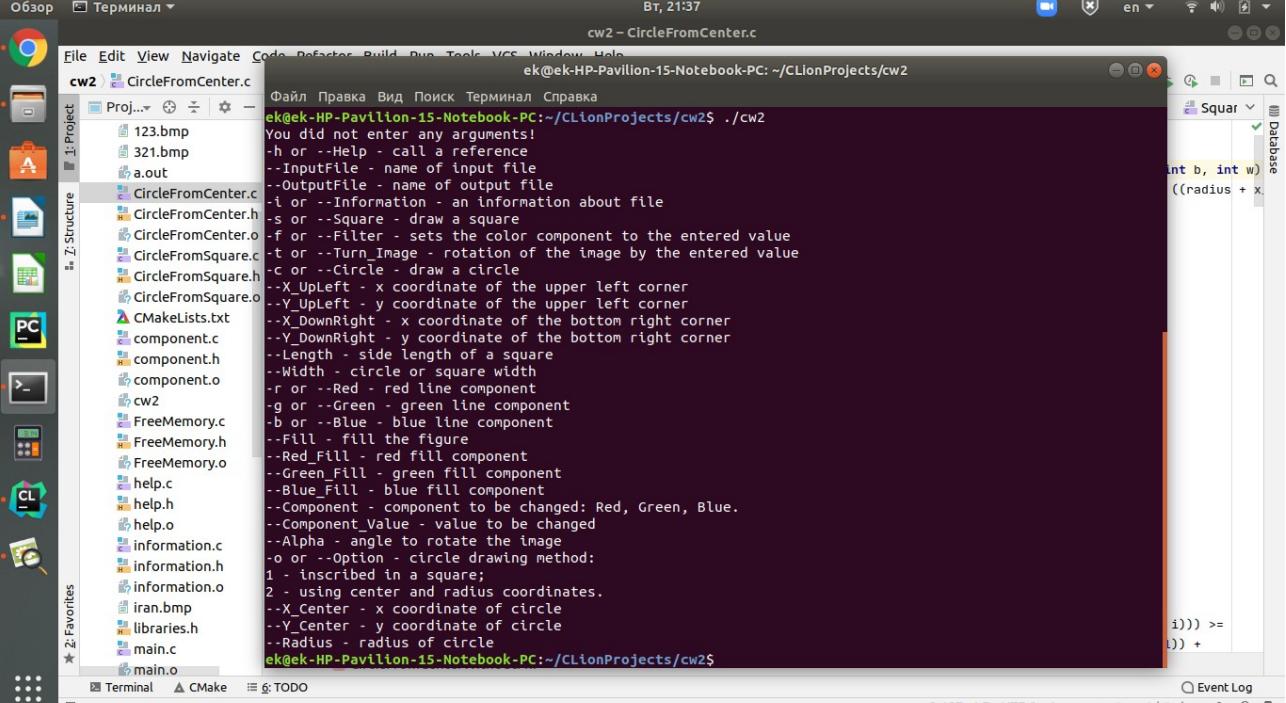
```
ek@ek-HP-Pavilion-15-Notebook-PC:~/CLionProjects/cw2$ ./cw2 -c -t -o 2 --InputFile iran.bmp --X_Center 100 --Y_Center 100 --Width 20 -r 255 -g 255 -b 255 --Radius 200 --OutputFile 12.bmp
```

The terminal then displays the help message for the program:

```
You called more than one function!  
-h or --Help - call a reference  
--InputFile - name of input file  
--OutputFile - name of output file  
-i or --Information - an information about file  
-s or --Square - draw a square  
-f or --Filter - sets the color component to the entered value  
-t or --Turn_Image - rotation of the image by the entered value  
-c or --Circle - draw a circle  
--X_UpLeft - x coordinate of the upper left corner  
--Y_UpLeft - y coordinate of the upper left corner  
--X_DownRight - x coordinate of the bottom right corner  
--Y_DownRight - y coordinate of the bottom right corner  
--Length - side length of a square  
--Width - circle or square width  
-r or --Red - red line component  
-g or --Green - green line component  
-b or --Blue - blue line component  
--Fill - fill the figure  
--Red_Fill - red fill component  
--Green_Fill - green fill component  
--Blue_Fill - blue fill component  
--Component - component to be changed: Red, Green, Blue.  
--Component_Value - value to be changed  
--Alpha - angle to rotate the image  
-o or --Option - circle drawing method:  
1 - inscribed in a square;  
2 - using center and radius coordinates.  
--X_Center - x coordinate of circle  
--Y_Center - y coordinate of circle  
--Radius - radius of circle
```

At the bottom of the terminal window, the prompt is shown again: `ek@ek-HP-Pavilion-15-Notebook-PC:~/CLionProjects/cw2$`.

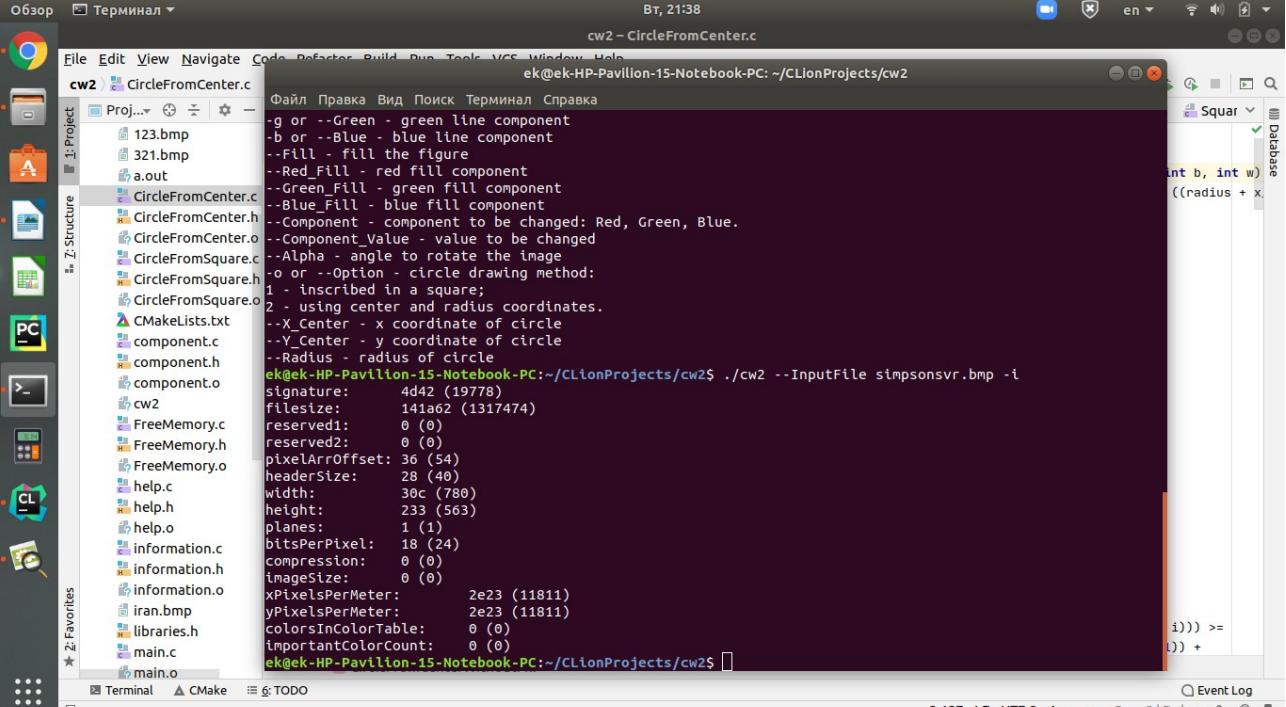
## Без аргументов



CLion IDE interface showing the terminal window with the command `./cw2`. The output displays usage information for the application:

```
You did not enter any arguments!
-h or --Help - call a reference
--Inputfile - name of input file
--Outputfile - name of output file
-i or --Information - an information about file
-s or --Square - draw a square
-f or --Filter - sets the color component to the entered value
-t or --Turn_Image - rotation of the image by the entered value
-c or --Circle - draw a circle
--X_UpperLeft - x coordinate of the upper left corner
--Y_UpperLeft - y coordinate of the upper left corner
--X_DownRight - x coordinate of the bottom right corner
--Y_DownRight - y coordinate of the bottom right corner
--Length - side length of a square
--Width - circle or square width
-r or --Red - red line component
-g or --Green - green line component
-b or --Blue - blue line component
--Fill - fill the figure
--Red_Fill - red fill component
--Green_Fill - green fill component
--Blue_Fill - blue fill component
--Component - component to be changed: Red, Green, Blue.
--Component_Value - value to be changed
--Alpha - angle to rotate the image
-o or --Option - circle drawing method:
1 - inscribed in a square;
2 - using center and radius coordinates.
--X_Center - x coordinate of circle
--Y_Center - y coordinate of circle
--Radius - radius of circle
```

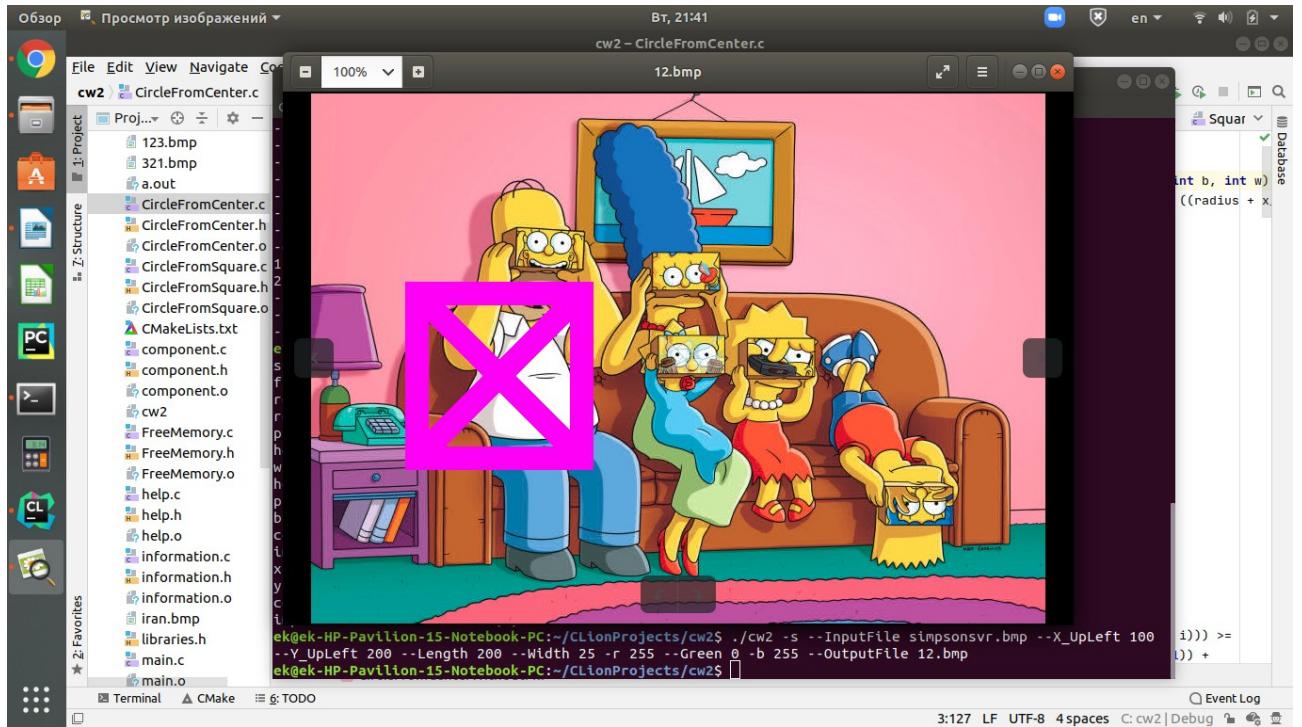
## Информация о файле



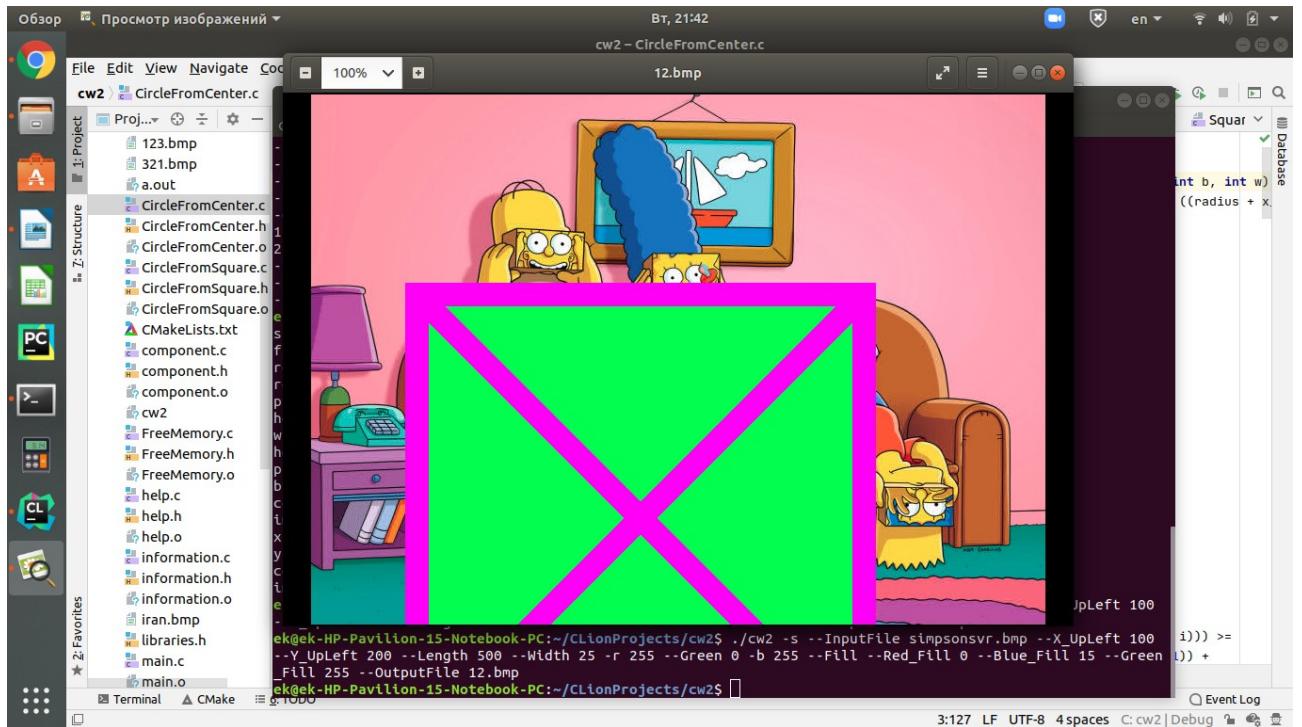
CLion IDE interface showing the terminal window with the command `./cw2 --InputFile simpsonsrv.bmp -i`. The output displays file metadata for the specified BMP image:

```
signature: 4d4d (19778)
filesize: 141a62 (1317474)
reserved1: 0 (0)
reserved2: 0 (0)
pixelAspectRatio: 36 (54)
headerSize: 28 (40)
width: 30c (780)
height: 233 (563)
planes: 1 (1)
bitsPerPixel: 18 (24)
compression: 0 (0)
imageSize: 0 (0)
xPixelsPerMeter: 2e23 (11811)
yPixelsPerMeter: 2e23 (11811)
colorsInColorTable: 0 (0)
importantColorCount: 0 (0)
```

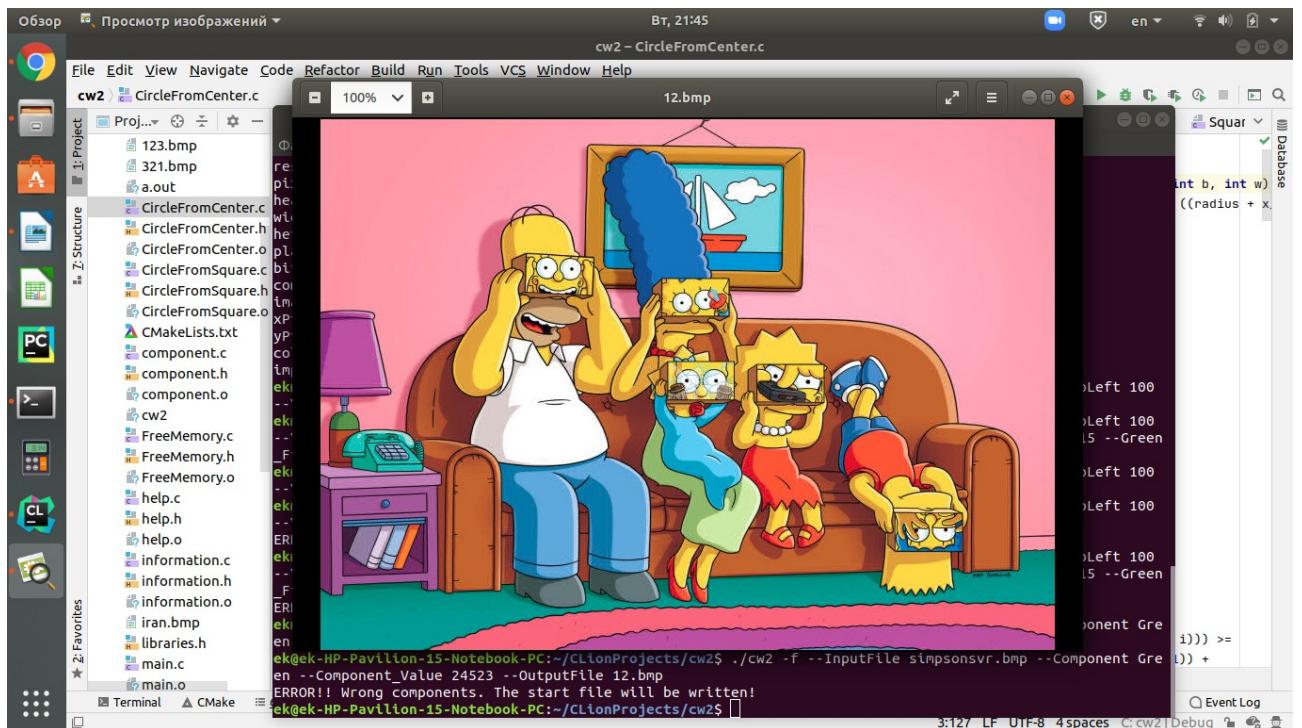
## Квадрат

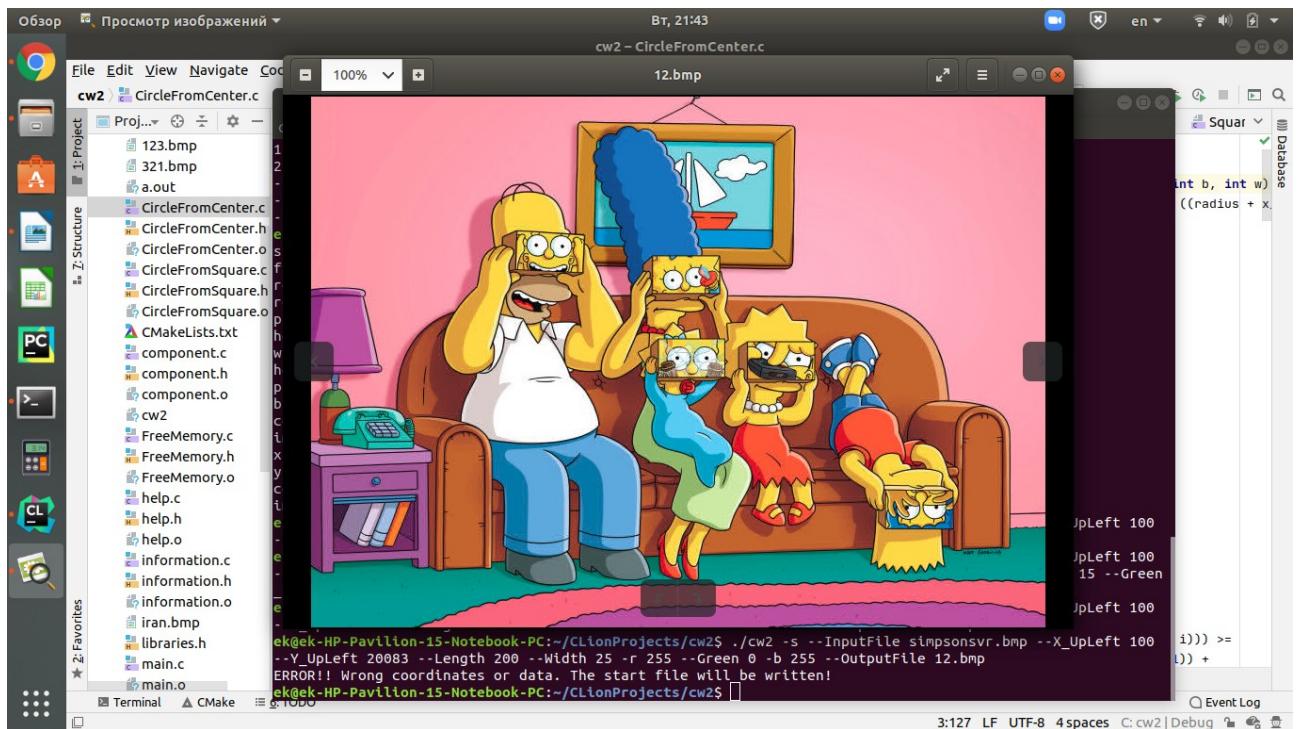
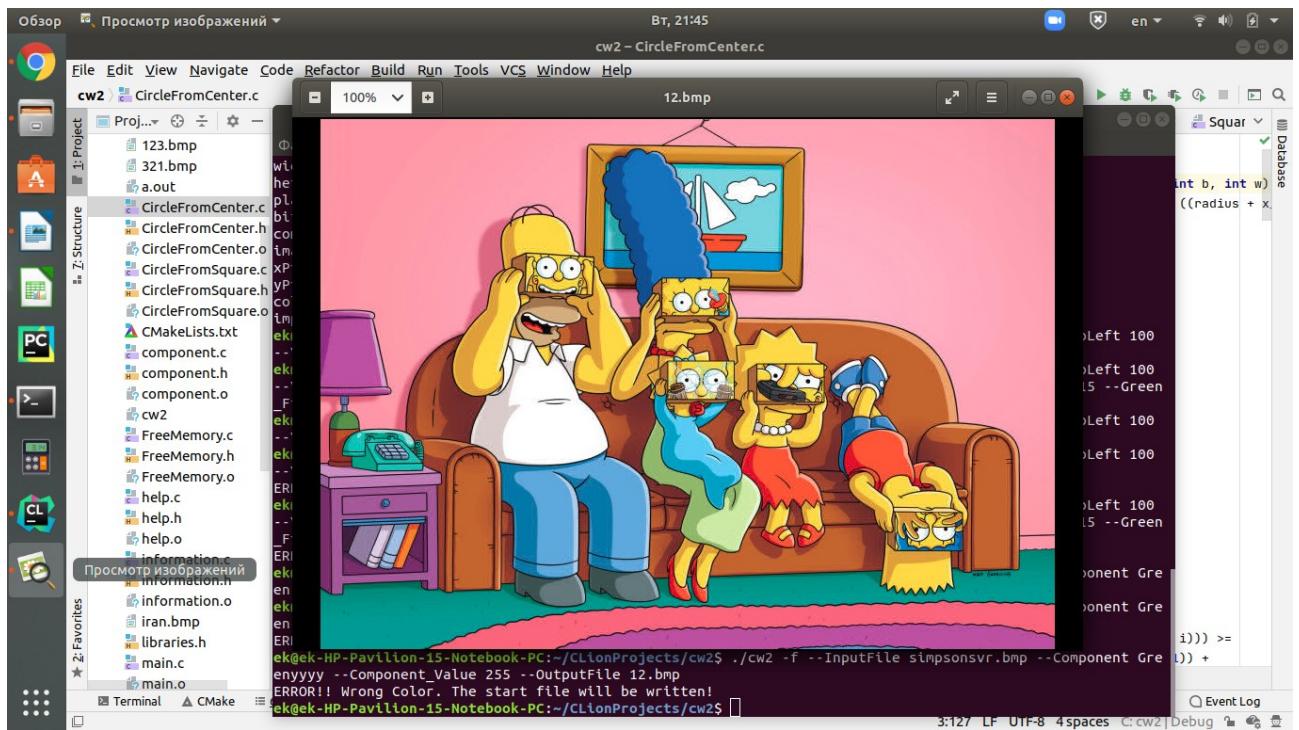


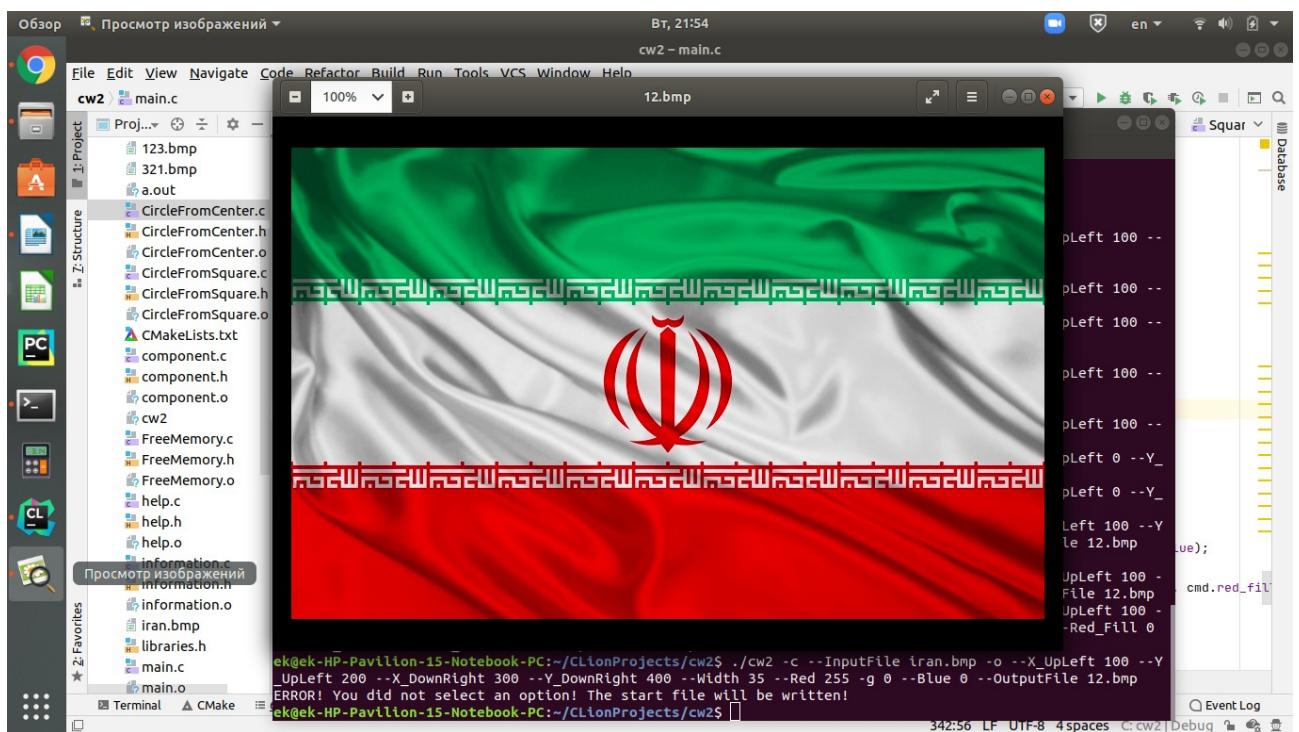
## Квадрат с заливкой



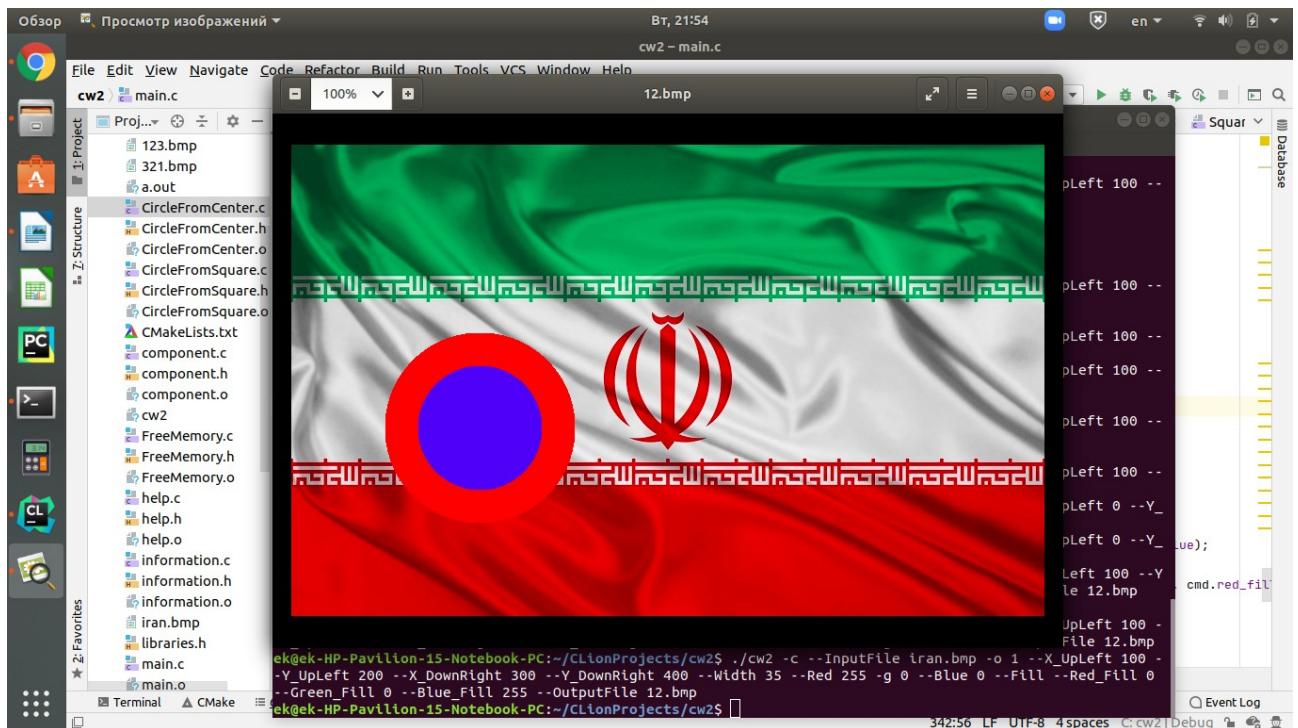
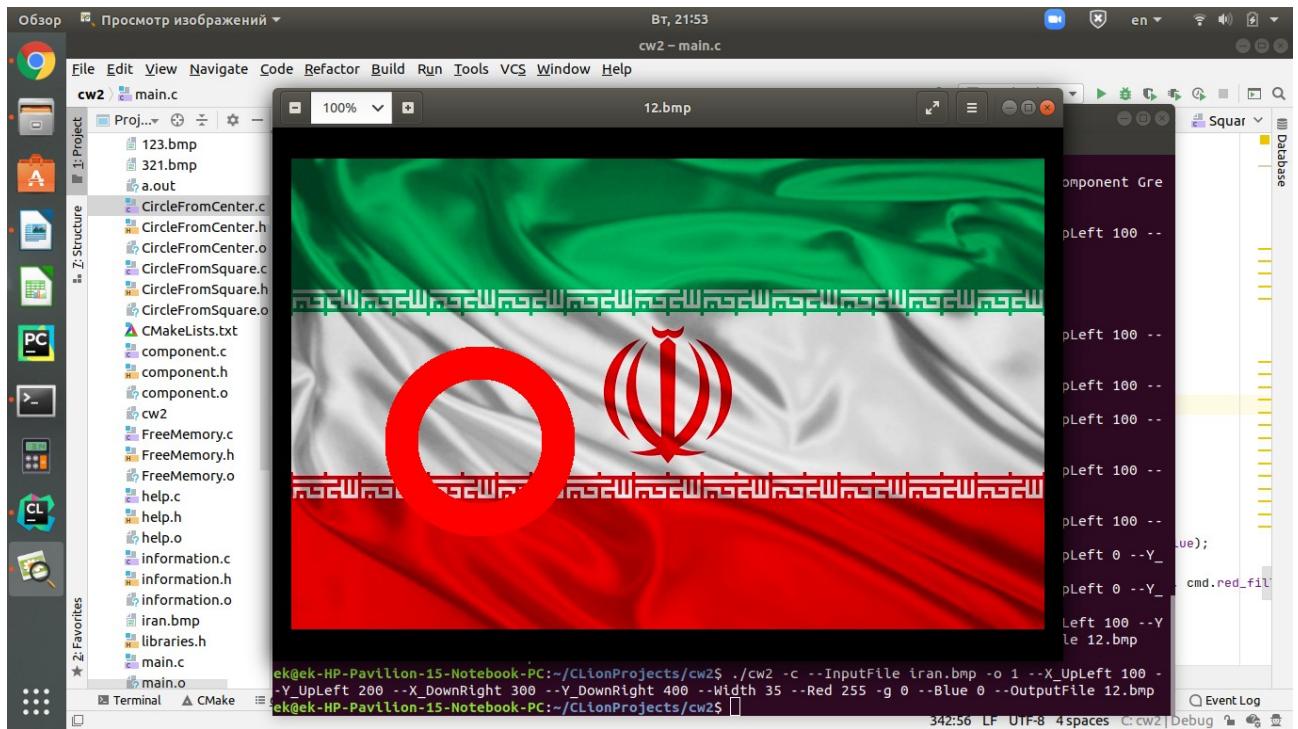
## Обработка ошибок



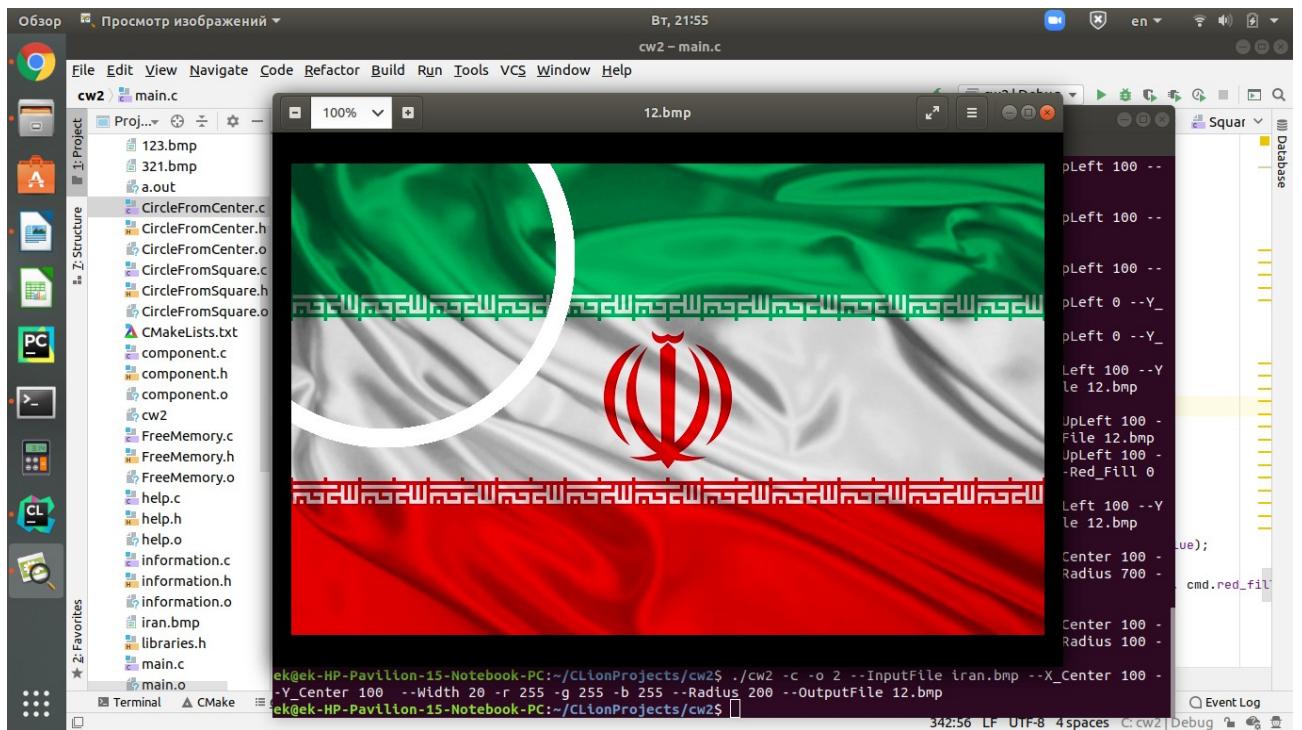
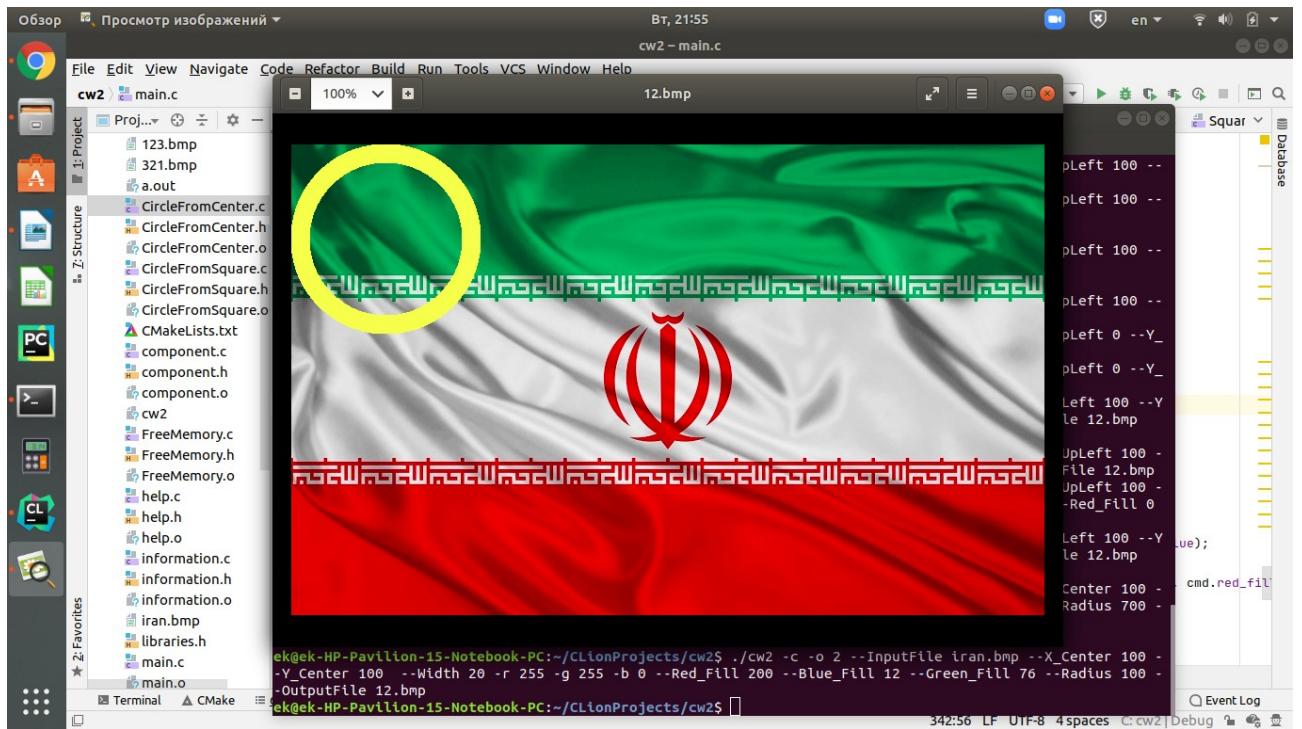




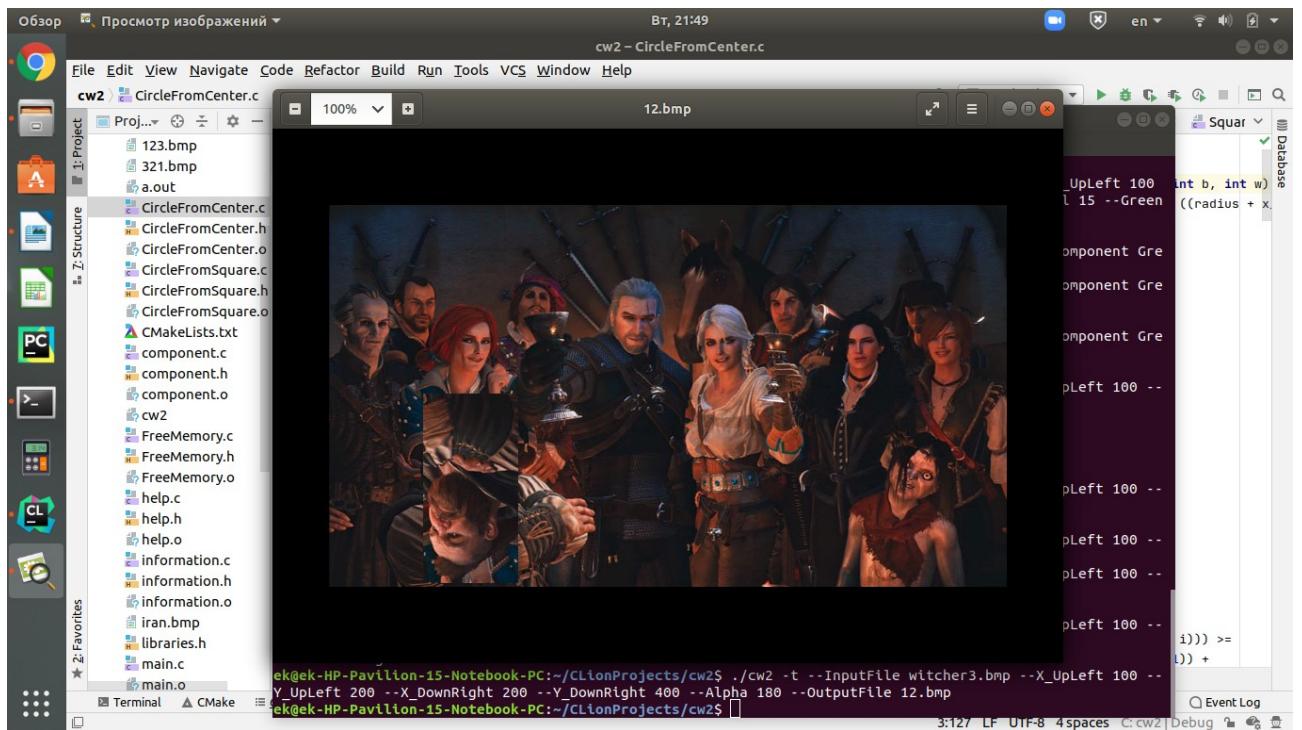
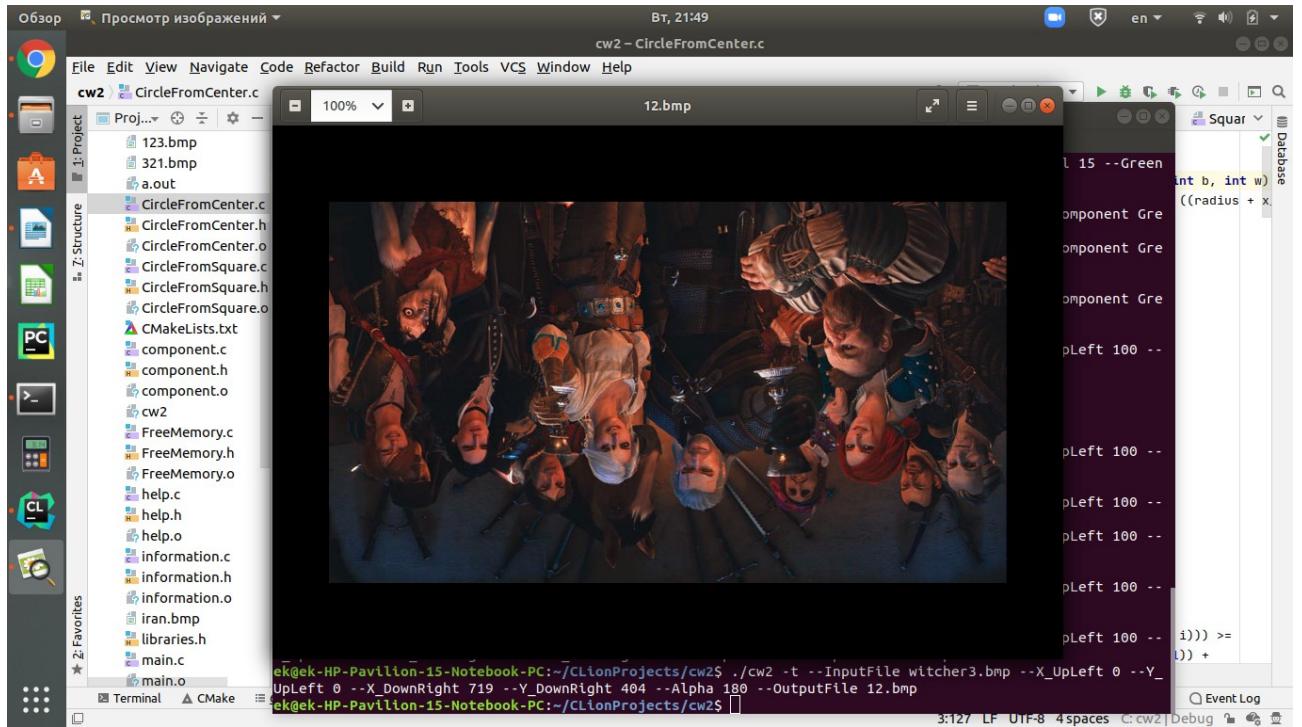
## Окружность по координатам квадрата

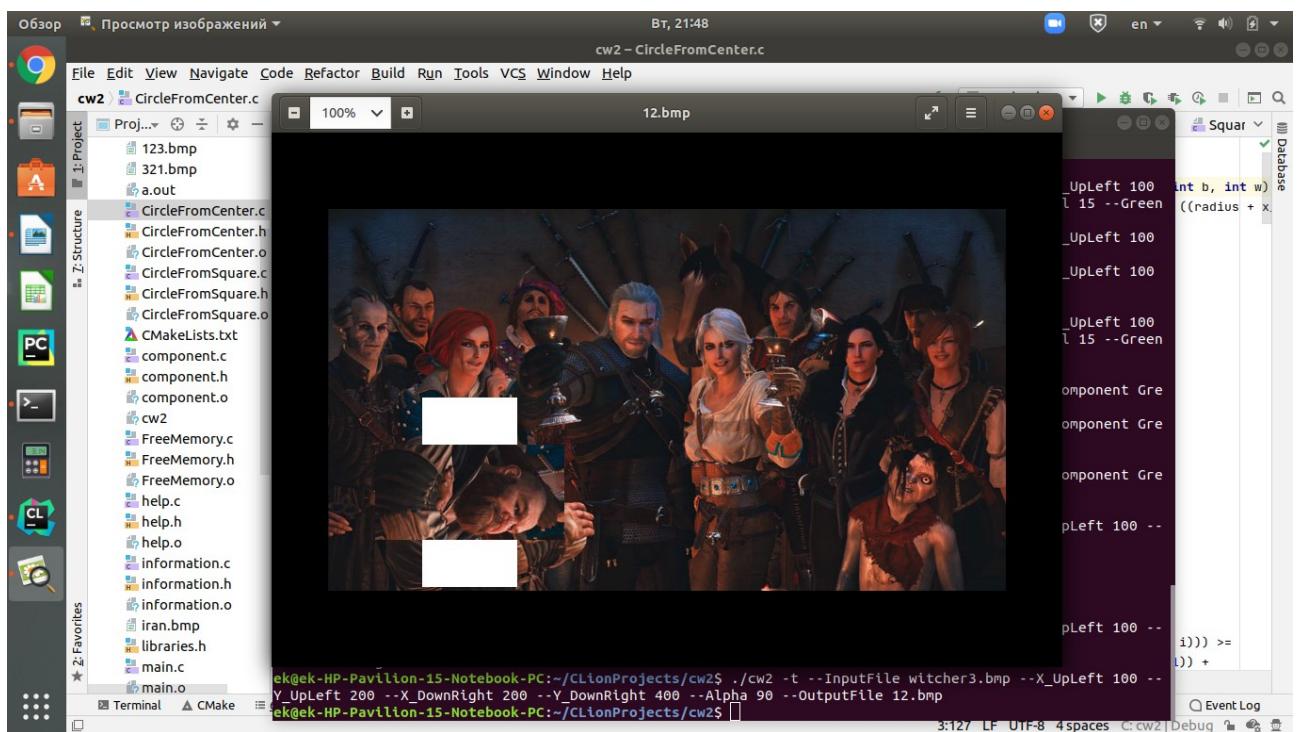
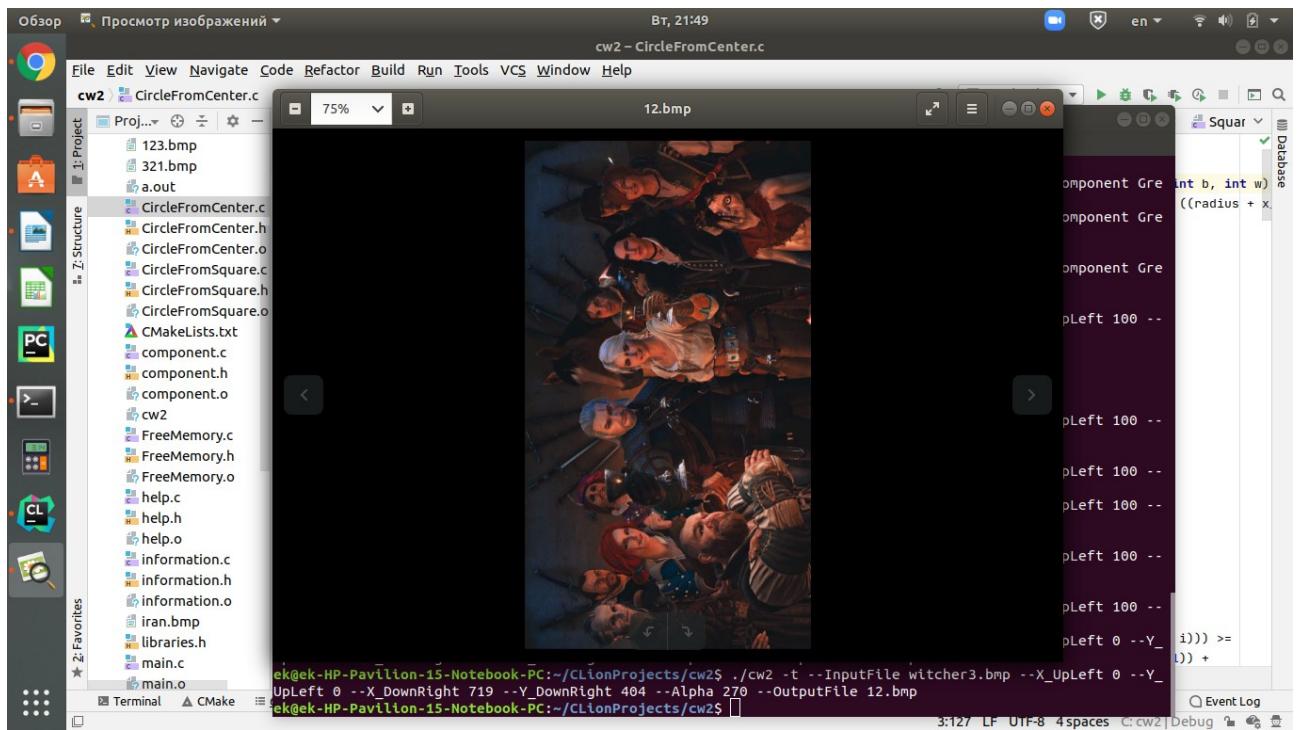


## Окружность по центру и радиусу

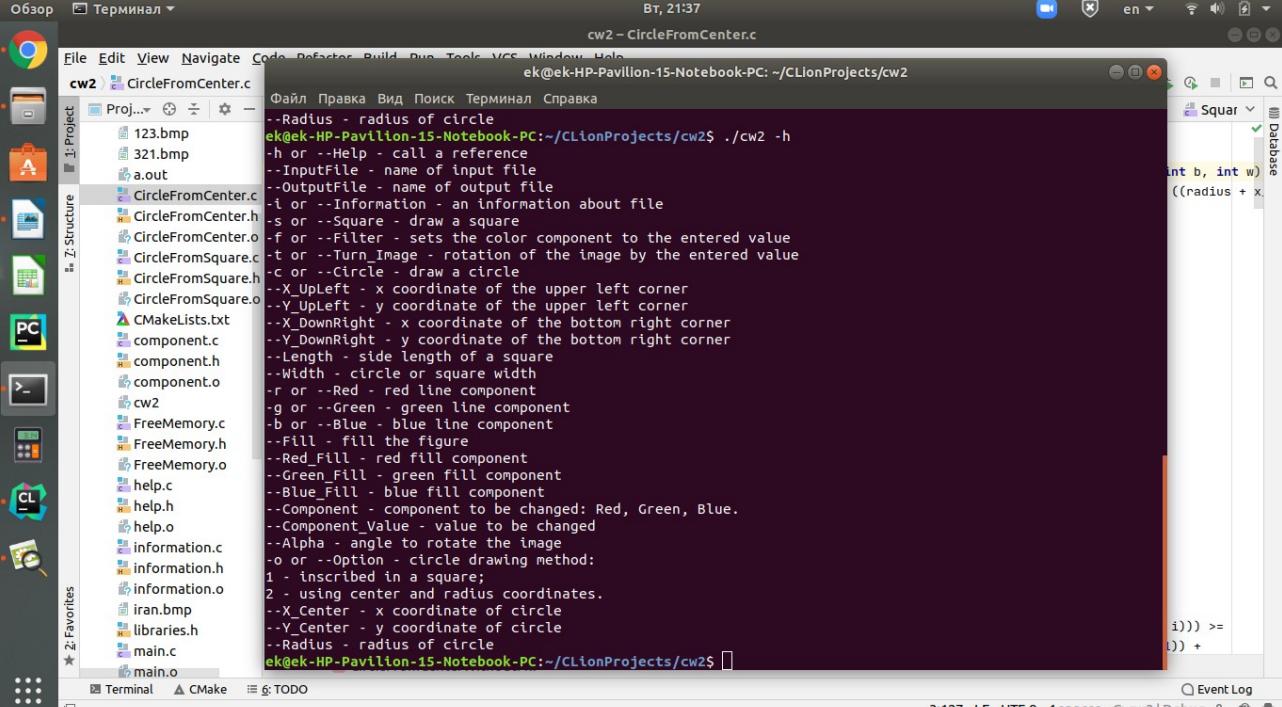


## Поворот изображения





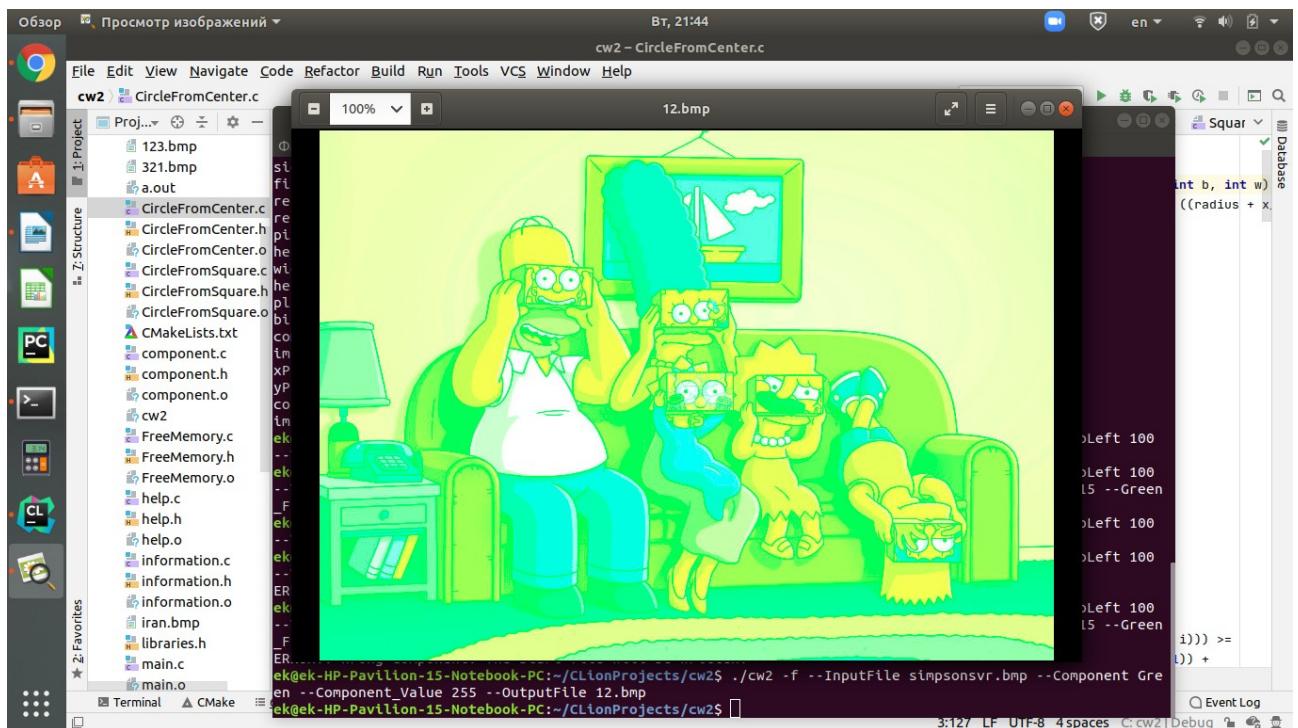
## Вызов помощи



The screenshot shows the CLion IDE interface with the terminal window open. The terminal shows the help output for the `./cw2 -h` command:

```
--Radius - radius of circle
-h or --Help - call a reference
--Inputfile - name of input file
--Outputfile - name of output file
-i or --Information - an information about file
-s or --Square - draw a square
-f or --Filter - sets the color component to the entered value
-t or --Turn_Image - rotation of the image by the entered value
-c or --Circle - draw a circle
--X_UpLeft - x coordinate of the upper left corner
--Y_UpLeft - y coordinate of the upper left corner
--X_DownRight - x coordinate of the bottom right corner
--Y_DownRight - y coordinate of the bottom right corner
--Length - side length of a square
--Width - circle or square width
--Red - red line component
--Green - green line component
--Blue - blue line component
--Fill - fill the figure
--Red_Fill - red fill component
--Green_Fill - green fill component
--Blue_Fill - blue fill component
--Component - component to be changed: Red, Green, Blue.
--Component_Value - value to be changed
--Alpha - angle to rotate the image
-o or --Option - circle drawing method:
1 - inscribed in a square;
2 - using center and radius coordinates.
--X_Center - x coordinate of circle
--Y_Center - y coordinate of circle
--Radius - radius of circle
```

## Фильтр



## **4. Заключение**

В результате выполнения курсовой работы получена программа, которая считывает введенное пользователем изображение формата bmp, динамически выделяет память под его хранение и обрабатывает его по выбору пользователя с помощью аргументов командной строки.

## **5. Использованные источники**

<http://www.cplusplus.com/>

<http://www.c-cpp.ru/>

<https://ru.wikipedia.org/wiki/>

## **Приложение А. Код программы**

`main.c`

```

#include "structures.h"
#include "libraries.h"
#include "component.h"
#include "CircleFromSquare.h"
#include "CircleFromCenter.h"
#include "Square.h"
#include "TurnImage.h"
#include "WriteImage.h"
#include "Swap.h"
#include "ReadImage.h"
#include "FreeMemory.h"
#include "help.h"
#include "information.h"

int main(int argc, char* argv[]) {
    int k;
    char inputfile[300];
    char outputfile[300];
    char *comp = calloc(100, sizeof(char));
    char find[11] = "1234567890";
    struct Commands cmd;
    cmd.n = 0;
    cmd.x1 = -1;
    cmd.y1 = -1;
    cmd.x2 = -1;
    cmd.y2 = -1;
    cmd.l = -1;
    cmd.w = -1;
    cmd.red = -1;
    cmd.green = -1;
    cmd.blue = -1;
    cmd.x_c = -1;
    cmd.y_c = -1;
    cmd.fill = 0;
    cmd.red_fill = -1;
    cmd.green_fill = -1;
    cmd.blue_fill = -1;
    cmd.alpha = -1;
    cmd.radius = -1;
}

```

```

cmd.component_value = -1;
cmd.option = -1;
cmd.error = -1;
cmd.output = 0;
int opt;
int longInd;
char* optStr = "r:g:b:o:sftchi";
struct option longOpts[]={
    {"Information", no_argument, NULL, 'i'},
    {"Help", no_argument, NULL, 'h'},
    {"InputFile", no_argument, NULL, 0},
    {"OutputFile", no_argument, NULL, 0},
    {"Square", no_argument, NULL, 's'},
    {"X_UpLeft", required_argument, NULL, 0},
    {"Y_UpLeft", required_argument, NULL, 0},
    {"X_DownRight", required_argument, NULL, 0},
    {"Y_DownRight", required_argument, NULL, 0},
    {"Length", required_argument, NULL, 0},
    {"Width", required_argument, NULL, 0},
    {"Red", required_argument, NULL, 'r'},
    {"Green", required_argument, NULL, 'g'},
    {"Blue", required_argument, NULL, 'b'},
    {"Fill", no_argument, NULL, 0},
    {"Red_Fill", required_argument, NULL, 0},
    {"Green_Fill", required_argument, NULL, 0},
    {"Blue_Fill", required_argument, NULL, 0},
    {"Filter", no_argument, NULL, 'f'},
    {"Component", required_argument, NULL, 0},
    {"Component_Value", required_argument, NULL, 0},
    {"Turn_Image", no_argument, NULL, 't'},
    {"Alpha", required_argument, NULL, 0},
    {"Circle", no_argument, NULL, 'c'},
    {"Option", required_argument, NULL, 'o'},
    {"X_Center", required_argument, NULL, 0},
    {"Y_Center", required_argument, NULL, 0},
    {"Radius", required_argument, NULL, 0},
    {NULL, 0, NULL, 0}
};

opt = getopt_long(argc, argv, optStr, longOpts, &longInd);

```

```

while(opt != -1) {
    switch (opt) {
        case 'r':
            k = 0;
            for(int i = 0; i < strlen(argv[optind - 1]); i++) {
                if(strchr(find, argv[optind - 1][i]) == NULL) {
                    k += 1;
                }
            }
            if (k == 0)
                cmd.red = atoi(argv[optind - 1]);
            break;
        case 'g':
            k = 0;
            for(int i = 0; i < strlen(argv[optind - 1]); i++) {
                if(strchr(find, argv[optind - 1][i]) == NULL) {
                    k += 1;
                }
            }
            if (k == 0)
                cmd.green = atoi(argv[optind - 1]);
            break;
        case 'b':
            k = 0;
            for(int i = 0; i < strlen(argv[optind - 1]); i++) {
                if(strchr(find, argv[optind - 1][i]) == NULL) {
                    k += 1;
                }
            }
            if (k == 0)
                cmd.blue = atoi(argv[optind - 1]);
            break;
        case 'o':
            k = 0;
            for(int i = 0; i < strlen(argv[optind - 1]); i++) {
                if(strchr(find, argv[optind - 1][i]) == NULL) {
                    k += 1;
                }
            }
    }
}

```

```

    if (k == 0)
        cmd.option = atoi(argv[optind - 1]);
    break;
case 's':
    cmd.n = 1;
    cmd.error += 1;
    break;
case 'f':
    cmd.n = 2;
    cmd.error += 1;
    break;
case 't':
    cmd.n = 3;
    cmd.error += 1;
    break;
case 'c':
    cmd.n = 4;
    cmd.error += 1;
    break;
case 'h':
    cmd.n = 5;
    cmd.error += 1;
    break;
case 'i':
    cmd.n = 6;
    cmd.error += 1;
    break;
case 0:
    if(strcmp("InputFile", longOpts[longInd].name)==0) {
        strcpy(inputfile, argv[optind]);
    }
    if(strcmp("OutputFile", longOpts[longInd].name)==0) {
        cmd.output = 1;
        strcpy(outputfile, argv[optind]);
    }
    if(strcmp("Fill", longOpts[longInd].name) == 0) {
        cmd.fill = 1;
    }
    if(strcmp("Component", longOpts[longInd].name) == 0) {

```

```

        strcpy(comp, argv[optind - 1]);
        comp[strlen(argv[optind - 1])] = '\0';
    }

    if(strcmp("Radius", longOpts[longInd].name) == 0) {
        k = 0;
        for(int i = 0; i < strlen(argv[optind - 1]); i++) {
            if(strchr(find, argv[optind - 1][i]) == NULL) {
                k += 1;
            }
        }
        if (k == 0)
            cmd.radius = atoi(argv[optind - 1]);
    }

    if(strcmp("Y_Center", longOpts[longInd].name)==0) {
        k = 0;
        for(int i = 0; i < strlen(argv[optind - 1]); i++) {
            if(strchr(find, argv[optind - 1][i]) == NULL) {
                k += 1;
            }
        }
        if (k == 0)
            cmd.y_c = atoi(argv[optind - 1]);
    }

    if(strcmp("X_Center", longOpts[longInd].name)==0) {
        k = 0;
        for(int i = 0; i < strlen(argv[optind - 1]); i++) {
            if(strchr(find, argv[optind - 1][i]) == NULL) {
                k += 1;
            }
        }
        if (k == 0)
            cmd.x_c = atoi(argv[optind - 1]);
    }

    if(strcmp("Alpha", longOpts[longInd].name)==0) {
        k = 0;
        for(int i = 0; i < strlen(argv[optind - 1]); i++) {
            if(strchr(find, argv[optind - 1][i]) == NULL) {
                k += 1;
            }
        }
    }

```

```

    }

    if (k == 0)
        cmd.alpha = atoi(argv[optind - 1]);
}

if(strcmp("Component_Value", longOpts[longInd].name)==0) {
    k = 0;
    for(int i = 0; i < strlen(argv[optind - 1]); i++) {
        if(strchr(find, argv[optind - 1][i]) == NULL) {
            k += 1;
        }
    }
    if (k == 0)
        cmd.component_value = atoi(argv[optind - 1]);
}

if(strcmp("X_UpLeft", longOpts[longInd].name)==0) {
    k = 0;
    for(int i = 0; i < strlen(argv[optind - 1]); i++) {
        if(strchr(find, argv[optind - 1][i]) == NULL) {
            k += 1;
        }
    }
    if (k == 0)
        cmd.x1 = atoi(argv[optind - 1]);
}

if(strcmp("X_DownRight", longOpts[longInd].name)==0) {
    k = 0;
    for(int i = 0; i < strlen(argv[optind - 1]); i++) {
        if(strchr(find, argv[optind - 1][i]) == NULL) {
            k += 1;
        }
    }
    if (k == 0)
        cmd.x2 = atoi(argv[optind - 1]);
}

if(strcmp("Y_DownRight", longOpts[longInd].name)==0) {
    k = 0;
    for(int i = 0; i < strlen(argv[optind - 1]); i++) {
        if(strchr(find, argv[optind - 1][i]) == NULL) {
            k += 1;
        }
    }
}

```

```

        }
    }

    if (k == 0)
        cmd.y2 = atoi(argv[optind - 1]);
    }

    if(strcmp("Y_UpLeft", longOpts[longInd].name)==0) {
        k = 0;
        for(int i = 0; i < strlen(argv[optind - 1]); i++) {
            if(strchr(find, argv[optind - 1][i]) == NULL) {
                k += 1;
            }
        }
        if (k == 0)
            cmd.y1 = atoi(argv[optind - 1]);
    }

    if(strcmp("Width", longOpts[longInd].name)==0) {
        k = 0;
        for(int i = 0; i < strlen(argv[optind - 1]); i++) {
            if(strchr(find, argv[optind - 1][i]) == NULL) {
                k += 1;
            }
        }
        if (k == 0)
            cmd.w = atoi(argv[optind - 1]);
    }

    if(strcmp("Length", longOpts[longInd].name)==0) {
        k = 0;
        for(int i = 0; i < strlen(argv[optind - 1]); i++) {
            if(strchr(find, argv[optind - 1][i]) == NULL) {
                k += 1;
            }
        }
        if (k == 0)
            cmd.l = atoi(argv[optind - 1]);
    }

    if(strcmp("Red_Fill", longOpts[longInd].name)==0) {
        k = 0;
        for(int i = 0; i < strlen(argv[optind - 1]); i++) {
            if(strchr(find, argv[optind - 1][i]) == NULL) {

```

```

        k += 1;
    }
}

if (k == 0)
    cmd.red_fill = atoi(argv[optind - 1]);
}

if(strcmp("Green_Fill", longOpts[longInd].name)==0) {
    k = 0;
    for(int i = 0; i < strlen(argv[optind - 1]); i++) {
        if(strchr(find, argv[optind - 1][i]) == NULL) {
            k += 1;
        }
    }
    if (k == 0)
        cmd.green_fill = atoi(argv[optind - 1]);
}

if(strcmp("Blue_Fill", longOpts[longInd].name)==0) {
    k = 0;
    for(int i = 0; i < strlen(argv[optind - 1]); i++) {
        if(strchr(find, argv[optind - 1][i]) == NULL) {
            k += 1;
        }
    }
    if (k == 0)
        cmd.blue_fill = atoi(argv[optind - 1]);
}

break;
}

opt = getopt_long(argc, argv, optStr, longOpts, &longInd);
}

if(argv[1]==NULL) {
    printf("You did not enter any arguments!\n");
    help();
    return 0;
}

if (cmd.error == -1) {
    printf("You did not call any function!\n");
    help();
}

```

```

        return 0;
    }

    else if(cmd.error != 0){
        printf("You called more than one function!\n");
        help();
        return 0;
    }

    if ((cmd.output == 0) && !((cmd.n == 5) || (cmd.n == 6))){
        printf("You did not enter an output file name!\n");
        help();
        return 0;
    }

    else if (cmd.n == 5){
        help();
        return 0;
    }

    FILE *file = fopen(inputfile, "rb");
    if (!file || (strstr(inputfile,".bmp") == NULL)) {
        printf("Could not open file.\n");
        return 0;
    }

    int result;

    BITMAPFILEHEADER *bmfh = calloc(1, sizeof(BITMAPFILEHEADER));
    BITMAPINFO *bmi = calloc(1, sizeof(BITMAPINFO));
    fread(bmfp, 1, sizeof(BITMAPFILEHEADER), file);
    fread(bmi, 1, sizeof(BITMAPINFO), file);
    if (bmi->size == 40){
        BITMAPINFOV3* bmiv3 = swap(bmi, file);
        if(bmiv3->bitCount != 24){
            printf("File format cannot be processed.");
            return 0;
        }
        RGB **image = ReadImage(bmiv3, file);
        switch (cmd.n){
            case 1:
                if (cmd.fill == 0)
                    MakeSquareWithoutFill(bmiv3, image, cmd.x1, cmd.y1,
                    cmd.l, cmd.w, cmd.red, cmd.green, cmd.blue);
                else

```

```

                MakeSquareWithFill(bmiv3, image, cmd.x1, cmd.y1,
cmd.l, cmd.w, cmd.red, cmd.green, cmd.blue, cmd.red_fill, cmd.green_fill,
cmd.blue_fill);
                break;
            case 2:
                component(bmiv3, image, comp, cmd.component_value);
                break;
            case 3:
                result = TurnImage(bmfh, bmiv3, image, cmd.x1, cmd.y1,
cmd.x2, cmd.y2, cmd.alpha, outputfile);
                if (result == 1){
                    return 0;
                }
                break;
            case 4:
                if(cmd.option == 1){
                    if (cmd.fill == 0)
                        CircleFromSquareWithoutFill(bmiv3, image, cmd.x1,
cmd.y1, cmd.x2, cmd.y2, cmd.red, cmd.green, cmd.blue, cmd.w);
                    else
                        CircleFromSquareWithFill(bmiv3, image, cmd.x1,
cmd.y1, cmd.x2, cmd.y2, cmd.red, cmd.green, cmd.blue, cmd.w,
cmd.red_fill, cmd.green_fill, cmd.blue_fill);
                }
                else{
                    if (cmd.option == 2){
                        if (cmd.fill == 0)
                            CircleFromCenterWithoutFill(bmiv3, image,
cmd.x_c, cmd.y_c, cmd.radius, cmd.red, cmd.green, cmd.blue, cmd.w);
                        else
                            CircleFromCenterWithFill(bmiv3, image,
cmd.x_c, cmd.y_c, cmd.radius, cmd.red, cmd.green, cmd.blue, cmd.w,
cmd.red_fill, cmd.green_fill, cmd.blue_fill);
                    }
                    else{
                        printf("ERROR! You did not select an option! The
start file will be written!\n");
                    }
                }
        }
    }
}

```

```

        break;

    case 6:
        printFileHeader(*bmfh);
        printInfoHeader(*bmiv3);
        break;
    default:
        printf("ERROR! You have not called any functions!\n");
        return 0;
    }

    WriteImage(bmfvh, bmiv3, image, outputfile);
    FreeMemory(image, bmiv3);
}

else{
    printf("File format cannot be processed.\n");
}

free(bmi);
free(bmfvh);
fclose(file);
return 0;
}

```

**CircleFromCenter.c**

```

#include "CircleFromCenter.h"

void CircleFromCenterWithoutFill(BITMAPINFOV3* bmiv3, RGB** image, int
x_c, int y_c, int radius, int r, int g, int b, int w) {
    if ((x_c < 0) || (x_c >= bmiv3->width) || (y_c < 0) || (y_c >= bmiv3-
>height) || (radius < 0) || (w < 0) || ((radius + x_c) >= bmiv3->width)
|| ((radius + x_c) >= bmiv3->width)) {
        printf("ERROR!! Wrong coordinates or data. The start file will be
written!\n");
    } else {
        if (w > radius) {
            printf("ERROR!! Width > Radius. The start file will be
written!\n");
        } else {
            if ((r < 0) || (r > 255) || (g < 0) || (g > 255) || (b < 0)
|| (b > 255)) {

```

```

        printf("ERROR!! Wrong components. The start file will be
written!\n");
    } else {
        int x1, x2, y1, y2;
        int y_tmp;
        int radius_tmp;
        x1 = x_c - radius;
        x2 = x_c + radius;
        y1 = y_c - radius;
        y2 = y_c + radius;
        for (int o = 0; o < w; o++) {
            radius_tmp = radius - o;
            for (int i = 0; i <= abs(y2 - y1) / 2; i++) {
                if (((int) bmiV3->height - y1 - 1 >= i) && (y1 +
i >= 0)) {
                    for (int j = 0; j <= abs(x2 - x1) / 2; j++) {
                        if ((x1 + j < bmiV3->width) && (x1 + j >=
0)) {
                            if (((((x_c - (x1 + j)) * (x_c - (x1 +
j)) + (y_c - (y1 + i)) * (y_c - (y1 + i))) >=
radius_tmp * radius_tmp) &&
(((x_c - (x1 + j + 1)) * (x_c - (x1 + j + 1)) +
(y_c - (y1 + i + 1)) * (y_c - (y1 + i + 1))) <
rad
ius_tmp * radius_tmp)) {
                                y_tmp = (int) bmiV3->height - y1
- 1;
                                image[y_tmp - i][x1 + j].r = r;
                                image[y_tmp - i][x1 + j].g = g;
                                image[y_tmp - i][x1 + j].b = b;
                            }
                        }
                    }
                if (x2 - j < bmiV3->width) {
                    if (((((x2 - j) - x_c) * ((x2 - j) -
x_c) + (y_c - (y1 + i)) * (y_c - (y1 + i))) >=
radius_tmp * radius_tmp) &&
(((x2 - j - 1) - x_c) * ((x2 - j - 1) - x_c) +

```

```

(y_
c - (y1 + i + 1)) * (y_c - (y1 + i + 1)) <
rad
ius_tmp * radius_tmp) {
y_tmp = (int) bmiV3->height - y1
- 1;
image[y_tmp - i][x2 - j].r = r;
image[y_tmp - i][x2 - j].g = g;
image[y_tmp - i][x2 - j].b = b;
}
}
}
}
for (int i = 0; i <= abs(y2 - y1) / 2; i++) {
if ((y2 >= i) && (y2 < bmiV3->height + i)) {
for (int j = 0; j <= abs(x2 - x1) / 2; j++) {
if ((x1 + j < bmiV3->width) && (x1 + j >=
0)) {
if (((x_c - (x1 + j)) * (x_c - (x1 +
j)) + ((y2 - i) - y_c) * ((y2 - i) - y_c)) >=
radius_tmp * radius_tmp) &&
(((x_c - (x1 + j + 1)) * (x_c - (x1 + j + 1)) +
((y2 - i - 1) - y_c) * ((y2 - i - 1) - y_c)) <
rad
ius_tmp * radius_tmp) {
y_tmp = (int) bmiV3->height - y2
- 1;
image[y_tmp + i][x1 + j].r = r;
image[y_tmp + i][x1 + j].g = g;
image[y_tmp + i][x1 + j].b = b;
}
}
if (x2 - j < bmiV3->width) {
if (((((x2 - j) - x_c) * ((x2 - j) -
x_c) + ((y2 - i) - y_c) * ((y2 - i) - y_c)) >=
radius_tmp * radius_tmp) &&
(((x2 - j - 1) - x_c) * ((x2 - j - 1) - x_c) +

```

```
void CircleFromCenterWithFill(BITMAPINFOV3* bmiV3, RGB** image, int x_c,
int y_c, int radius, int r, int g, int b, int w, int r_z, int g_z, int
b_z) {
    if ((x_c < 0) || (x_c >= bmiV3->width) || (y_c < 0) || (y_c >= bmiV3-
>height) || (radius < 0) || (w < 0) || ((radius + x_c) >= bmiV3->width)
|| ((radius + x_c) >= bmiV3->width)) {
        printf("ERROR!! Wrong coordinates or data. The start file will be
written!\n");
    } else {
        if (w > radius) {
            printf("ERROR!! Width > Radius. The start file will be
written!\n");
        } else {
            if ((r < 0) || (r > 255) || (g < 0) || (g > 255) || (b < 0)
|| (b > 255) || (r_z < 0) || (r_z > 255) || (g_z < 0) || (g_z > 255) ||
(b_z < 0) || (b_z > 255)) {
                printf("ERROR!! Wrong components. The start file will be
written!\n");
            }
        }
    }
}
```

```

} else {
    int x1, x2, y1, y2;
    int y_tmp;
    int radius_tmp;
    x1 = x_c - radius;
    x2 = x_c + radius;
    y1 = y_c - radius;
    y2 = y_c + radius;
    for (int o = 0; o < w; o++) {
        radius_tmp = radius - o;
        for (int i = 0; i <= abs(y2 - y1) / 2; i++) {
            if (((int) bmidV3->height - y1 - 1) >= i) && (y1 +
i >= 0)) {
                for (int j = 0; j <= abs(x2 - x1) / 2; j++) {
                    if ((x1 + j < bmidV3->width) && (x1 + j) >=
0)) {
                        if (((((x_c - (x1 + j)) * (x_c - (x1 +
j)) + (y_c - (y1 + i)) * (y_c - (y1 + i))) >=
radius_tmp * radius_tmp) &&
(((x_c - (x1 + j + 1)) * (x_c - (x1 + j + 1)) +
(y_c - (y1 + i + 1)) * (y_c - (y1 + i + 1))) <
rad
ius_tmp * radius_tmp)) {
                            y_tmp = (int) bmidV3->height - y1
- 1;
                            image[y_tmp - i][x1 + j].r = r;
                            image[y_tmp - i][x1 + j].g = g;
                            image[y_tmp - i][x1 + j].b = b;
                        }
                        if (((((x_c - (x1 + j)) * (x_c - (x1 +
j)) + (y_c - (y1 + i)) * (y_c - (y1 + i))) <
radius_tmp * radius_tmp) {
                            image[y_tmp - i][x1 + j].r = r_z;
                            image[y_tmp - i][x1 + j].g = g_z;
                            image[y_tmp - i][x1 + j].b = b_z;
                        }
                    }
                    if (x2 - j < bmidV3->width) {

```

```

        if (((((x2 - j) - x_c) * ((x2 - j) -
x_c) + (y_c - (y1 + i)) * (y_c - (y1 + i))) >=
radius_tmp * radius_tmp) &&
(((x2 - j - 1) - x_c) * ((x2 - j - 1) - x_c) +
(y_c - (y1 + i + 1)) * (y_c - (y1 + i + 1)) <
rad
ius_tmp * radius_tmp) ) {
    y_tmp = (int) bmiV3->height - y1
- 1;
    image[y_tmp - i][x2 - j].r = r;
    image[y_tmp - i][x2 - j].g = g;
    image[y_tmp - i][x2 - j].b = b;
}
if (((((x2 - j) - x_c) * ((x2 - j) -
x_c) + (y_c - (y1 + i)) * (y_c - (y1 + i))) <
radius_tmp * radius_tmp) {
    image[y_tmp - i][x2 - j].r = r_z;
    image[y_tmp - i][x2 - j].g = g_z;
    image[y_tmp - i][x2 - j].b = b_z;
}
}
}
}
}
for (int i = 0; i <= abs(y2 - y1) / 2; i++) {
    if ((y2 >= i) && (y2 < bmiV3->height + i)) {
        for (int j = 0; j <= abs(x2 - x1) / 2; j++) {
            if ((x1 + j < bmiV3->width) && (x1 + j >=
0)) {
                if (((((x_c - (x1 + j)) * (x_c - (x1 +
j)) + ((y2 - i) - y_c) * ((y2 - i) - y_c)) >=
radius_tmp * radius_tmp) &&
(((x_c - (x1 + j + 1)) * (x_c - (x1 + j + 1)) +
(y2 - i - 1) - y_c) * ((y2 - i - 1) - y_c)) <
rad
ius_tmp * radius_tmp) ) {

```

```

y_tmp = (int) bmiV3->height - y2
- 1;

image[y_tmp + i][x1 + j].r = r;
image[y_tmp + i][x1 + j].g = g;
image[y_tmp + i][x1 + j].b = b;
}

if (((x_c - (x1 + j)) * (x_c - (x1 +
j)) + ((y2 - i) - y_c) * ((y2 - i) - y_c)) <
    radius_tmp * radius_tmp) {
    image[y_tmp + i][x1 + j].r = r_z;
    image[y_tmp + i][x1 + j].g = g_z;
    image[y_tmp + i][x1 + j].b = b_z;
}
}

if (x2 - j < bmiV3->width) {
    if (((((x2 - j) - x_c) * ((x2 - j) -
x_c) + ((y2 - i) - y_c) * ((y2 - i) - y_c)) >=
        radius_tmp * radius_tmp) &&
(((x2 - j - 1) - x_c) * ((x2 - j - 1) - x_c) +
(y2 - i - 1) - y_c) * ((y2 - i - 1) - y_c)) <
    rad
ius_tmp * radius_tmp)) {
    y_tmp = (int) bmiV3->height - y2
- 1;

image[y_tmp + i][x2 - j].r = r;
image[y_tmp + i][x2 - j].g = g;
image[y_tmp + i][x2 - j].b = b;
}

if (((((x2 - j) - x_c) * ((x2 - j) -
x_c) + ((y2 - i) - y_c) * ((y2 - i) - y_c)) <
    radius_tmp * radius_tmp) {
    image[y_tmp + i][x2 - j].r = r_z;
    image[y_tmp + i][x2 - j].g = g_z;
    image[y_tmp + i][x2 - j].b = b_z;
}
}
}
}
}
```

## CircleFromCenter.h

```
#ifndef CIRCLEFROMCENTER_H
#define CIRCLEFROMCENTER_H

#include "libraries.h"
#include "structures.h"

void CircleFromCenterWithoutFill(BITMAPINFOV3* bmiV3, RGB** image, int x_c, int y_c, int radius, int r, int g, int b, int w);
void CircleFromCenterWithFill(BITMAPINFOV3* bmiV3, RGB** image, int x_c, int y_c, int radius, int r, int g, int b, int w, int r_z, int g_z, int b_z);

#endif
```

## CircleFromSquare.c

```

#include "CircleFromSquare.h"

void CircleFromSquareWithoutFill(BITMAPINFOV3* bmiV3, RGB** image ,int
x1, int y1, int x2, int y2, int r, int g, int b, int d) {
    if ((x1 < 0) || (x1 >= bmiV3->width) || (y1 < 0) || (y1 >= bmiV3-
>height) || (x2 < 0) || (x2 >= bmiV3->width) ||
(y2 < 0) || (y2 >= bmiV3->height) || (x2 < x1) || (y2 < y1) || (d
< 0)) {
        printf("ERROR!! Wrong coordinates or data. The start file will be
written!\n");
    } else {
        if (d > (x2 - x1) / 2) {
            printf("ERROR!! Width > Radius. The start file will be
written!\n");
        } else {

```

```

        if ((r < 0) || (r > 255) || (g < 0) || (g > 255) || (b < 0)
|| (b > 255)) {
            printf("ERROR!! Wrong components. The start file will be
written!\n");
        } else {
            if ((x2 - x1) != (y2 - y1)) {
                printf("ERROR!! Don't square. The start file will be
written!\n");
            } else {
                int x_c, y_c;
                int radius;
                int y_tmp;
                x_c = (int) ((x1 + x2) / 2);
                y_c = (int) ((y1 + y2) / 2);
                for (int o = 0; o < d; o++) {
                    radius = (int) ((x2 - x1) / 2) - o;
                    for (int i = 0; i <= abs(y2 - y1) / 2; i++) {
                        if (((int) bmiV3->height - y1 - 1) >= i) {
                            for (int j = 0; j <= abs(x2 - x1) / 2; j++)
+)
{
                            if (x1 + j < bmiV3->width) {
                                if (((((x_c - (x1 + j)) * (x_c -
(x1 + j)) +
(y_c - (y1 + i)) * (y_c -
(y1 + i))) >=
radius * radius) && (((x_c -
(x1 + j + 1)) * (x_c - (x1 + j + 1)) +
(y_c - (y1 + i + 1)) * (y_c - (y1 + i + 1))) <
radius *
radius))
{
                                y_tmp = (int) bmiV3->height -
y1 - 1;
                                image[y_tmp - i][x1 + j].r =
r;
                                image[y_tmp - i][x1 + j].g =
g;
                                image[y_tmp - i][x1 + j].b =
b;

```

```

        }
    }

    if (x2 - j < bmiV3->width) {
        if (((((x2 - j) - x_c) * ((x2 -
j) - x_c) +
(y_c - (y1 + i)) * (y_c -
(y1 + i))) >=
radius * radius) && (((x2 -
j - 1) - x_c) * ((x2 - j - 1) - x_c) +
(y_c - (y1 + i + 1)) * (y_c - (y1 + i + 1)) <
radius *
radius) {
            y_tmp = (int) bmiV3->height -
y1 - 1;
            image[y_tmp - i][x2 - j].r =
r;
            image[y_tmp - i][x2 - j].g =
g;
            image[y_tmp - i][x2 - j].b =
b;
        }
    }
}
}

}

for (int i = 0; i <= abs(y2 - y1) / 2; i++) {
    if ((y2 >= i) && (y2 < bmiV3->height + i)) {
        for (int j = 0; j <= abs(x2 - x1) / 2; j++)
        {
            if (x1 + j < bmiV3->width) {
                if (((((x_c - (x1 + j)) * (x_c -
(x1 + j)) +
((y2 - i) - y_c) * ((y2 -
i) - y_c)) >=
radius * radius) && (((x_c -
(x1 + j + 1)) * (x_c - (x1 + j + 1)) +
((y2 - i - 1) - y_c) * ((y2 - i - 1) - y_c)) <
radius * radius)

```



```

        }
    }

}

void CircleFromSquareWithFill(BITMAPINFOV3 *bmiV3, RGB **image, int x1,
int y1, int x2, int y2, int r, int g, int b,
int d, int r_z, int g_z, int b_z) {
    if ((x1 < 0) || (x1 >= bmiV3->width) || (y1 < 0) || (y1 >= bmiV3-
>height) || (x2 < 0) || (x2 >= bmiV3->width) ||
(y2 < 0) || (y2 >= bmiV3->height) || (x2 < x1) || (y2 < y1) || (d
< 0)) {
        printf("ERROR!! Wrong coordinates or data. The start file will be
written!\n");
    } else {
        if (d > (x2 - x1) / 2) {
            printf("ERROR!! Width > Radius. The start file will be
written!\n");
        } else {
            if ((r < 0) || (r > 255) || (g < 0) || (g > 255) || (b < 0)
|| (b > 255) || (r_z < 0) || (r_z > 255) ||
(g_z < 0) || (g_z > 255) || (b_z < 0) || (b_z > 255)) {
                printf("ERROR!! Wrong components. The start file will be
written!\n");
            } else {
                if ((x2 - x1) != (y2 - y1)) {
                    printf("ERROR!! Don't square. The start file will be
written!\n");
                } else {
                    int x_c, y_c;
                    int radius;
                    int y_tmp;
                    x_c = (int) ((x1 + x2) / 2);
                    y_c = (int) ((y1 + y2) / 2);
                    for (int o = 0; o < d; o++) {
                        radius = (int) ((x2 - x1) / 2) - o;
                        for (int i = 0; i <= abs(y2 - y1) / 2; i++) {
                            if ((int) bmiV3->height - y1 - 1 >= i) {
                                for (int j = 0; j <= abs(x2 - x1) / 2; j+
+) {

```

```

        if (x1 + j < bmiV3->width) {
            if (((x_c - (x1 + j)) * (x_c -
(x1 + j)) +
(y_c - (y1 + i)) * (y_c -
(y1 + i))) >=
radius * radius) && (((x_c -
(x1 + j + 1)) * (x_c - (x1 + j + 1)) +
(y_c - (y1 + i + 1)) * (y_c - (y1 + i + 1))) <
radius
* radius)) {
                y_tmp = (int) bmiV3->height -
y1 - 1;
                image[y_tmp - i][x1 + j].r =
r;
                image[y_tmp - i][x1 + j].g =
g;
                image[y_tmp - i][x1 + j].b =
b;
            }
            if (((x_c - (x1 + j)) * (x_c -
(x1 + j)) +
(y_c - (y1 + i)) * (y_c -
(y1 + i))) <
radius * radius) {
                image[y_tmp - i][x1 + j].r =
r_z;
                image[y_tmp - i][x1 + j].g =
g_z;
                image[y_tmp - i][x1 + j].b =
b_z;
            }
        }
        if (x2 - j < bmiV3->width) {
            if (((((x2 - j) - x_c) * ((x2 -
j) - x_c) +
(y_c - (y1 + i)) * (y_c -
(y1 + i))) >=

```

```

radius * radius) && (((x2 -
j - 1) - x_c) * ((x2 - j - 1) - x_c) +
(y1 + i + 1)) * (y_c - (y1 + i + 1)) <
radius
* radius)) {
y_tmp = (int) bmiV3->height -
y1 - 1;
image[y_tmp - i][x2 - j].r =
r;
image[y_tmp - i][x2 - j].g =
g;
image[y_tmp - i][x2 - j].b =
b;
}
if (((((x2 - j) - x_c) * ((x2 - j) -
x_c) +
(y_c - (y1 + i)) * (y_c -
(y1 + i))) <
radius * radius) {
image[y_tmp - i][x2 - j].r =
r_z;
image[y_tmp - i][x2 - j].g =
g_z;
image[y_tmp - i][x2 - j].b =
b_z;
}
}
}
}
for (int i = 0; i <= abs(y2 - y1) / 2; i++) {
if ((y2 >= i) && (y2 < bmiV3->height + i)) {
for (int j = 0; j <= abs(x2 - x1) / 2; j+
+) {
if (x1 + j < bmiV3->width) {
if (((((x_c - (x1 + j)) * (x_c -
(x1 + j)) +

```

```

((y2 - i) - y_c) * ((y2 -
i) - y_c)) >=
radius * radius) && (((x_c -
(x1 + j + 1)) * (x_c - (x1 + j + 1)) +
((y2 - i - 1) - y_c) * ((y2 - i - 1) - y_c)) <
radius *
radius)) {
y_tmp = (int) bmiV3->height -
y2 - 1;
image[y_tmp + i][x1 + j].r =
r;
image[y_tmp + i][x1 + j].g =
g;
image[y_tmp + i][x1 + j].b =
b;
}
if (((x_c - (x1 + j)) * (x_c -
(x1 + j)) +
((y2 - i) - y_c) * ((y2 - i) -
y_c)) <
radius * radius) {
image[y_tmp + i][x1 + j].r =
r_z;
image[y_tmp + i][x1 + j].g =
g_z;
image[y_tmp + i][x1 + j].b =
b_z;
}
if (x2 - j < bmiV3->width) {
if (((((x2 - j) - x_c) * ((x2 -
j) - x_c)) +
((y2 - i) - y_c) * ((y2 -
i) - y_c)) >=
radius * radius) && (((((x2 -
j - 1) - x_c) * ((x2 - j - 1) - x_c) +
((y2 - i - 1) - y_c) * ((y2 - i - 1) - y_c)) <

```

## CircleFromSquare.h

```
#ifndef CIRCLEFROMSQUARE_H  
#define CIRCLEFROMSQUARE_H
```

```

#include "libraries.h"
#include "structures.h"

void CircleFromSquareWithoutFill(BITMAPINFOV3* bmiV3, RGB** image ,int
x1, int y1, int x2, int y2, int r, int g, int b, int d);
void CircleFromSquareWithFill(BITMAPINFOV3* bmiV3, RGB** image ,int x1,
int y1, int x2, int y2, int r, int g, int b, int d, int r_z, int g_z, int
b_z);

#endif

```

### **component.c**

```

#include "component.h"

void component(BITMAPINFOV3* bmiV3, RGB** image, char* s, int
component_value){

    if ((component_value < 0) || (component_value > 255)){
        printf("ERROR!! Wrong components. The start file will be
written!\n");
    }
    else {
        if (strcmp(s, "Red") == 0){
            for (int i = 0; i < bmiV3->height; i++)
                for (int j = 0; j < bmiV3->width; j++)
                    image[i][j].r = component_value;
        }
        else{
            if (strcmp(s, "Green") == 0){
                for (int i = 0; i < bmiV3->height; i++)
                    for (int j = 0; j < bmiV3->width; j++)
                        image[i][j].g = component_value;
            }
            else{
                if (strcmp(s, "Blue") == 0){
                    for (int i = 0; i < bmiV3->height; i++)
                        for (int j = 0; j < bmiV3->width; j++)
                            image[i][j].b = component_value;
                }
            }
        }
    }
}

```

```
    printf("ERROR!! Wrong Color. The start file will be
written!\n");
}
}
}
}
}
}
```

## component.h

```
#ifndef COMPONENT_H
#define COMPONENT_H

#include "structures.h"
#include "libraries.h"

void component(BITMAPINFOV3* bmiV3, RGB** image, char* s, int
component_value);

#endif
```

FreeMemory.c

```
#include "FreeMemory.h"

void FreeMemory(RGB** image, BITMAPINFOV3* bmiV3) {
    for(int i = 0; i < bmiV3->height; i++)
        free(image[i]);
    free(image);
    free(bmiV3);
}
```

## FreeMemory.h

```
#ifndef FREEMEMORY_H
#define FREEMEMORY_H

#include "structures.h"
#include "libraries.h"
void FreeMemory(RGB** image, BITMAPINFOV3* bmiV3);

#endif
```

**help.c**

```
#include "help.h"

void help(){
    printf("-h or --Help - call a reference\n");
    printf("--InputFile - name of input file\n");
    printf("--OutputFile - name of output file\n");
    printf("-i or --Information - an information about file\n");
    printf("-s or --Square - draw a square\n");
    printf("-f or --Filter - sets the color component to the entered
value\n");
    printf("-t or --Turn_Image - rotation of the image by the entered
value\n");
    printf("-c or --Circle - draw a circle\n");
    printf("--X_UpLeft - x coordinate of the upper left corner\n");
    printf("--Y_UpLeft - y coordinate of the upper left corner\n");
    printf("--X_DownRight - x coordinate of the bottom right corner\n");
    printf("--Y_DownRight - y coordinate of the bottom right corner\n");
    printf("--Length - side length of a square\n");
    printf("--Width - circle or square width\n");
    printf("-r or --Red - red line component\n");
    printf("-g or --Green - green line component\n");
    printf("-b or --Blue - blue line component\n");
    printf("--Fill - fill the figure\n");
    printf("--Red_Fill - red fill component\n");
    printf("--Green_Fill - green fill component\n");
    printf("--Blue_Fill - blue fill component\n");
    printf("--Component - component to be changed: Red, Green, Blue.\n");
    printf("--Component_Value - value to be changed\n");
    printf("--Alpha - angle to rotate the image\n");
    printf("-o or --Option - circle drawing method:\n");
    printf("1 - inscribed in a square;\n");
    printf("2 - using center and radius coordinates.\n");
    printf("--X_Center - x coordinate of circle\n");
    printf("--Y_Center - y coordinate of circle\n");
    printf("--Radius - radius of circle\n");
}
```

**help.h**

```

#ifndef HELP_H
#define HELP_H

#include "libraries.h"
void help();

#endif

information.c
#include "information.h"

void printFileHeader(BITMAPFILEHEADER header) {
    printf("signature:\t%x (%hu)\n", header.type, header.type);
    printf("filesize:\t%x (%u)\n", header.size, header.size);
    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.offBits, header.offBits);
}

void printInfoHeader(BITMAPINFOV3 header) {
    printf("headerSize:\t%x (%u)\n", header.size, header.size);
    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.bitCount,
header.bitCount);
    printf("compression:\t%x (%u)\n", header.compression,
header.compression);
    printf("imageSize:\t%x (%u)\n", header.sizeImage, header.sizeImage);
    printf("xPixelsPerMeter:\t%x (%u)\n", header.XPelsPerMeter,
header.XPelsPerMeter);
    printf("yPixelsPerMeter:\t%x (%u)\n", header.YPelsPerMeter,
header.YPelsPerMeter);
    printf("colorsInColorTable:\t%x (%u)\n", header.clrUsed,
header.clrUsed);
    printf("importantColorCount:\t%x (%u)\n", header.clrImportant,
header.clrImportant);
}

```

```

information.h

#ifndef INFORMATION_H
#define INFORMATION_H

#include "structures.h"
#include "libraries.h"
void printFileHeader(BITMAPFILEHEADER header);
void printInfoHeader(BITMAPINFOV3 header);

#endif

```

```

libraries.h

#ifndef LIBRARIES_H
#define LIBRARIES_H

```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#include <getopt.h>
#include <unistd.h>
#include <string.h>

```

```

#endif

```

### **ReadImage.c**

```

#include "ReadImage.h"

RGB** ReadImage(BITMAPINFOV3* bmiV3, FILE* file){

    RGB **image = malloc(bmiV3->height * sizeof(RGB*));
    for (int i = 0; i < bmiV3->height; i++){
        image[i] = malloc(bmiV3->width * sizeof(RGB) + (bmiV3->width) % 4);
        fread(image[i], bmiV3->width * sizeof(RGB) + (bmiV3->width) % 4,
        1, file);
    }
    return image;
}

```

```
ReadImage.h
#ifndef CW2_READIMAGE_H
#define CW2_READIMAGE_H

#include "structures.h"
#include "libraries.h"
RGB** ReadImage(BITMAPINFOV3* bmiV3, FILE* file);

#endif
```

### **Square.c**

```
#include "Square.h"

void MakeSquareWithoutFill(BITMAPINFOV3* bmiV3, RGB** image, int x, int
y, int l, int w, int r, int g, int b){
    if ((x >= bmiV3->width) || (y >= bmiV3->height) || (x < 0) || (y < 0)
    || (l < 0) || (w < 0))
        printf("ERROR!! Wrong coordinates or data. The start file will be
written!\n");
    else if ((r < 0) || (r > 255) || (g < 0) || (g > 255) || (b < 0) ||
(b > 255)){
        printf("ERROR!! Wrong component. The start file will be written!\n");
    }
    else if (w > l / 2){
        printf("ERROR!! Width > side. The start file will be written!\n");
    }
    else{
        y = (int)bmiV3->height - 1 - y;
        for (int p = 0; p < w; p++){
            if (y >= p){
                for (int v = 0; v < l; v++){
                    if (x + v < bmiV3->width){
                        image[y - p][x + v].r = r;
                        image[y - p][x + v].g = g;
                        image[y - p][x + v].b = b;
                    }
                }
            }
        }
    }
}
```

```

        }

    }

for(int f = 0; f < (l - 2 * w); f++) {
    if (y >= f + w) {
        for(int k = 0; k < w; k++) {
            if (x + k < bmiV3->width) {
                image[y - w - f][x + k].r = r;
                image[y - w - f][x + k].g = g;
                image[y - w - f][x + k].b = b;
            }
            if (x + l - k - 1 < bmiV3->width) {
                image[y - w - f][x + l - k - 1].r = r;
                image[y - w - f][x + l - k - 1].g = g;
                image[y - w - f][x + l - k - 1].b = b;
            }
        }
    }
}

for (int p = 0; p < w; p++) {
    if (y >= l - w + p) {
        for (int v = 0; v < l; v++) {
            if (x + v < bmiV3->width) {
                image[y - (l - w + p)][x + v].r = r;
                image[y - (l - w + p)][x + v].g = g;
                image[y - (l - w + p)][x + v].b = b;
            }
        }
    }
}

int size = (int)(w / sqrt(2));
for (int j = w; j < l - w; j++) {
    for(int m = 0; m < size + 1; m++) {
        if ((y >= j + m) && (x + j < bmiV3->width)) {
            image[y - j - m][x + j].r = r;
            image[y - j - m][x + j].g = g;
            image[y - j - m][x + j].b = b;
        }
        if ((y >= j - m) && (x + j < bmiV3->width)) {

```

```

        image[y - j + m][x + j].r = r;
        image[y - j + m][x + j].g = g;
        image[y - j + m][x + j].b = b;
    }

    if ((y >= j - m) && (x + l - j - 1 < bmiV3->width)) {
        image[y - j + m][x + l - j - 1].r = r;
        image[y - j + m][x + l - j - 1].g = g;
        image[y - j + m][x + l - j - 1].b = b;
    }

    if ((y >= j + m) && (x + l - j - 1 < bmiV3->width)) {
        image[y - j - m][x + l - j - 1].r = r;
        image[y - j - m][x + l - j - 1].g = g;
        image[y - j - m][x + l - j - 1].b = b;
    }
}

}

void MakeSquareWithFill(BITMAPINFOV3* bmiV3, RGB** image, int x, int y,
int l, int w, int r, int g, int b, int r_z, int g_z, int b_z){
    if ((x >= bmiV3->width) || (y >= bmiV3->height) || (x < 0) || (y < 0)
|| (l < 0) || (w < 0))
        printf("ERROR!! Wrong coordinates or data. The start file will be
written!\n");
    else if ((r < 0) || (r > 255) || (g < 0) || (g > 255) || (b < 0) ||
(b > 255) || (r_z < 0) || (r_z > 255) || (g_z < 0) || (g_z > 255) || (b_z
< 0) || (b_z > 255)){
        printf("ERROR!! Wrong component. The start file will be written!\n");
    }
    else if (w > l / 2){
        printf("ERROR!! Width > side. The start file will be written!\n");
    }
    else{
        y = (int)bmiV3->height - 1 - y;
        for (int p = 0; p < w; p++){
            if (y >= p){

```

```

        for (int v = 0; v < l; v++) {
            if (x + v < bmiV3->width) {
                image[y - p][x + v].r = r;
                image[y - p][x + v].g = g;
                image[y - p][x + v].b = b;
            }
        }
    }

    for(int f = 0; f < (l - 2 * w); f++) {
        if (y >= f + w) {
            for(int k = 0; k < w; k++) {
                if (x + k < bmiV3->width) {
                    image[y - w - f][x + k].r = r;
                    image[y - w - f][x + k].g = g;
                    image[y - w - f][x + k].b = b;
                }
                if (x + l - k - 1 < bmiV3->width) {
                    image[y - w - f][x + l - k - 1].r = r;
                    image[y - w - f][x + l - k - 1].g = g;
                    image[y - w - f][x + l - k - 1].b = b;
                }
            }
        }
        for(int a = 0; a < l - 2 * w; a++) {
            if (x + w + a < bmiV3->width) {
                image[y - w - f][x + w + a].r = r_z;
                image[y - w - f][x + w + a].g = g_z;
                image[y - w - f][x + w + a].b = b_z;
            }
        }
    }

    for (int p = 0; p < w; p++) {
        if (y >= l - w + p) {
            for (int v = 0; v < l; v++) {
                if (x + v < bmiV3->width) {
                    image[y - (l - w + p)][x + v].r = r;
                    image[y - (l - w + p)][x + v].g = g;
                    image[y - (l - w + p)][x + v].b = b;
                }
            }
        }
    }
}

```

```

    }

}

int size = (int)(w / sqrt(2));
for (int j = w; j < l - w; j++) {
    for(int m = 0; m < size + 1; m++) {
        if ((y >= j + m) && (x + j < bmiV3->width)) {
            image[y - j - m][x + j].r = r;
            image[y - j - m][x + j].g = g;
            image[y - j - m][x + j].b = b;
        }
        if ((y >= j - m) && (x + j < bmiV3->width)) {
            image[y - j + m][x + j].r = r;
            image[y - j + m][x + j].g = g;
            image[y - j + m][x + j].b = b;
        }
        if ((y >= j - m) && (x + l - j - 1 < bmiV3->width)) {
            image[y - j + m][x + l - j - 1].r = r;
            image[y - j + m][x + l - j - 1].g = g;
            image[y - j + m][x + l - j - 1].b = b;
        }
        if ((y >= j + m) && (x + l - j - 1 < bmiV3->width)) {
            image[y - j - m][x + l - j - 1].r = r;
            image[y - j - m][x + l - j - 1].g = g;
            image[y - j - m][x + l - j - 1].b = b;
        }
    }
}
}
}
```

## Square.h

```
#ifndef SQUARE_H  
#define SQUARE_H
```

```
#include "structures.h"
```

```

#include "libraries.h"

void MakeSquareWithoutFill(BITMAPINFOV3* bmiV3, RGB** image, int x, int
y, int l, int d, int r, int g, int b);
void MakeSquareWithFill(BITMAPINFOV3* bmiV3, RGB** image, int x, int y,
int l, int w, int r, int g, int b, int r_z, int g_z, int b_z);

#endif

structures.h

#ifndef STRUCTURES_H
#define STRUCTURES_H

#pragma pack(push, 1)

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

typedef struct{
    unsigned short type;
    unsigned int size;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int offBits;
}BITMAPFILEHEADER;

typedef struct {
    unsigned int size;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bitCount;
    unsigned int compression;
    unsigned int sizeImage;
    unsigned int XPelsPerMeter;
    unsigned int YPelsPerMeter;
    unsigned int clrUsed;
    unsigned int clrImportant;
}

```

```

}BITMAPINFOV3;

typedef struct {
    unsigned int size;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bitCount;
}BITMAPINFO;

typedef struct {
    unsigned int compression;
    unsigned int sizeImage;
    unsigned int XPelsPerMeter;
    unsigned int YPelsPerMeter;
    unsigned int clrUsed;
    unsigned int clrImportant;
}BITMAPINFOV3_tmp;
#pragma pack(pop)

struct Commands{
    int n;
    int x_c;
    int y_c;
    int x1;
    int y1;
    int x2;
    int y2;
    int radius;
    int l;
    int w;
    int red;
    int green;
    int blue;
    int fill;
    int red_fill;
    int green_fill;
    int blue_fill;
    int alpha;
}

```

```

    int component_value;
    int option;
    int error;
    int output;
};

}
```

### **Swap.c**

```
#include "Swap.h"

BITMAPINFOV3* swap(BITMAPINFO* bmi, FILE* file){
    BITMAPINFOV3_tmp *bmiv3_tmp = calloc(1, sizeof(BITMAPINFOV3_tmp));
    BITMAPINFOV3 *bmiv3 = calloc(1, sizeof(BITMAPINFOV3));
    fread(bmiv3_tmp, 1, sizeof(BITMAPINFOV3_tmp), file);
    bmiv3->size = bmi->size;
    bmiv3->width = bmi->width;
    bmiv3->height = bmi->height;
    bmiv3->planes = bmi->planes;
    bmiv3->bitCount = bmi->bitCount;
    bmiv3->compression = bmiv3_tmp->compression;
    bmiv3->sizeImage = bmiv3_tmp->sizeImage;
    bmiv3->XPelsPerMeter = bmiv3_tmp->XPelsPerMeter;
    bmiv3->YPelsPerMeter = bmiv3_tmp->YPelsPerMeter;
    bmiv3->clrUsed = bmiv3_tmp->clrUsed;
    bmiv3->clrImportant = bmiv3_tmp->clrImportant;
    free(bmiv3_tmp);
    return bmiv3;
}
```

### **Swap.h**

```
#ifndef CW2_SWAP_H
#define CW2_SWAP_H

#include "structures.h"
#include "libraries.h"
BITMAPINFOV3* swap(BITMAPINFO* bmi, FILE* file);

#endif
```

### **TurnImage.c**

```

#include "TurnImage.h"

int TurnImage(BITMAPFILEHEADER* bmfh, BITMAPINFOV3* bmiV3, RGB** image,
int x1, int y1, int x2, int y2, int alpha, char* outputfile) {
    if ((x1 >= bmiV3->width) || (y1 >= bmiV3->height) || (x1 < 0) || (y1
< 0) || (x2 < 0) || (x2 >= bmiV3->width) || (y2 < 0) || (y2 >= bmiV3-
>height) || (x2 < x1) || (y2 < y1))
        printf("ERROR!! Wrong coordinates or data. The start file will be
written!\n");
    else if ((alpha != 90) && (alpha != 180) && (alpha != 270))
        printf("ERROR!! Wrong angle. The start file will be written!\n");
    else {
        int x_c, y_c;
        int cur;
        int start_x;
        unsigned int width_tmp;
        RGB *tmp;
        RGB **tmp_image;
        if (((x1 == 0) && (x2 == bmiV3->width - 1) && (y1 == 0) && (y2 ==
bmiV3->height - 1)) &&
            ((alpha == 90) || (alpha == 270))) {
            tmp_image = malloc(bmiV3->width * sizeof(RGB *));
            for (int i = 0; i < abs(x2 - x1) + 1; i++)
                tmp_image[i] = malloc(bmiV3->height * sizeof(RGB));
            y2 = (int) bmiV3->height - 1 - y2;
            y1 = (int) bmiV3->height - 1 - y1;
            if (alpha == 90) {
                for (int i = 0; i < bmiV3->width; i++) {
                    for (int j = 0; j < bmiV3->height; j++) {
                        tmp_image[i][j].r = image[y2 + j][x2 - i].r;
                        tmp_image[i][j].g = image[y2 + j][x2 - i].g;
                        tmp_image[i][j].b = image[y2 + j][x2 - i].b;
                    }
                }
            } else {
                for (int i = 0; i < abs(x2 - x1) + 1; i++) {
                    for (int j = 0; j < abs(y2 - y1) + 1; j++) {
                        tmp_image[i][j].r = image[y1 - j][x1 + i].r;
                        tmp_image[i][j].g = image[y1 - j][x1 + i].g;

```

```

        tmp_image[i][j].b = image[y1 - j][x1 + i].b;
    }
}

width_tmp = bmiV3->width;
bmiV3->width = bmiV3->height;
bmiV3->height = width_tmp;
WriteImage(bmfh, bmiV3, tmp_image, outputfile);
for (int i = 0; i < abs(x2 - x1) + 1; i++)
    free(tmp_image[i]);
free(tmp_image);
return 1;
} else {
    if ((alpha == 90) || (alpha == 270)) {
        x_c = (int) ((x2 + x1) / 2);
        y_c = (int) ((y2 + y1) / 2);
        if ((x2 - x1) % 2 == 0)
            cur = (int) y_c - (int) ((x2 - x1) / 2);
        else
            cur = (int) y_c - (int) ((x2 - x1 + 1) / 2);
        cur = (int) bmiV3->height - 1 - cur;
        if (abs(y2 - y1) % 2 == 0) {
            start_x = x_c - (int) (abs(y2 - y1) / 2);
        } else
            start_x = x_c - (int) (abs(y2 - y1 + 1) / 2);
        y2 = (int) bmiV3->height - 1 - y2;
        y1 = (int) bmiV3->height - 1 - y1;
        tmp_image = malloc((abs(x2 - x1) + 1) * sizeof(RGB *));
        for (int i = 0; i < abs(x2 - x1) + 1; i++)
            tmp_image[i] = malloc((abs(y2 - y1) + 1) *
sizeof(RGB));
        if (alpha == 90) {
            for (int i = 0; i < abs(x2 - x1) + 1; i++) {
                for (int j = 0; j < abs(y2 - y1) + 1; j++) {
                    tmp_image[i][j].r = image[y2 + j][x1 + i].r;
                    tmp_image[i][j].g = image[y2 + j][x1 + i].g;
                    tmp_image[i][j].b = image[y2 + j][x1 + i].b;
                }
            }
        }
    }
}

```

```

        if ((x2 - x1) > abs(y2 - y1)) {
            for (int i = 0; i < (x2 - x1 - abs(y2 - y1)) / 2;
i++) {
                for (int j = 0; j <= abs(y2 - y1); j++) {
                    image[y1 - j][x_c + ((int) (abs(y2 -
y1) / 2) + i)].r = 255;
                    image[y1 - j][x_c + ((int) (abs(y2 -
y1) / 2) + i)].g = 255;
                    image[y1 - j][x_c + ((int) (abs(y2 -
y1) / 2) + i)].b = 255;
                    image[y1 - j][x_c - ((int) (abs(y2 -
y1) / 2) + i)].r = 255;
                    image[y1 - j][x_c - ((int) (abs(y2 -
y1) / 2) + i)].g = 255;
                    image[y1 - j][x_c - ((int) (abs(y2 -
y1) / 2) + i)].b = 255;
                }
            }
        } else if ((x2 - x1) < abs(y2 - y1)) {
            for (int i = 0; i <= (abs(y2 - y1) - (x2 - x1)) /
2; i++) {
                for (int j = 0; j <= abs(x2 - x1); j++) {
                    image[bmiV3->height - y_c - 1 + ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].r = 255;
                    image[bmiV3->height - y_c - 1 + ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].g = 255;
                    image[bmiV3->height - y_c - 1 + ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].b = 255;
                    image[bmiV3->height - y_c - 1 - ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].r = 255;
                    image[bmiV3->height - y_c - 1 - ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].g = 255;
                    image[bmiV3->height - y_c - 1 - ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].b = 255;
                }
            }
        }
    }
}

```

```

        for (int i = 0; i < abs(x2 - x1) + 1; i++) {
            for (int j = 0; j < abs(y2 - y1) + 1; j++) {
                tmp_image[i][j].r = image[y1 - j][x2 - i].r;
                tmp_image[i][j].g = image[y1 - j][x2 - i].g;
                tmp_image[i][j].b = image[y1 - j][x2 - i].b;
            }
        }

        if ((x2 - x1) > abs(y2 - y1)) {
            for (int i = 0; i < (x2 - x1 - abs(y2 - y1)) / 2;
i++) {
                for (int j = 0; j <= abs(y2 - y1); j++) {
                    image[y1 - j][x_c + ((int) (abs(y2 -
y1) / 2) + i)].r = 255;
                    image[y1 - j][x_c + ((int) (abs(y2 -
y1) / 2) + i)].g = 255;
                    image[y1 - j][x_c + ((int) (abs(y2 -
y1) / 2) + i)].b = 255;
                    image[y1 - j][x_c - ((int) (abs(y2 -
y1) / 2) + i)].r = 255;
                    image[y1 - j][x_c - ((int) (abs(y2 -
y1) / 2) + i)].g = 255;
                    image[y1 - j][x_c - ((int) (abs(y2 -
y1) / 2) + i)].b = 255;
                }
            }
        } else if ((x2 - x1) < abs(y2 - y1)) {
            for (int i = 0; i <= (abs(y2 - y1) - (x2 - x1)) /
2; i++) {
                for (int j = 0; j <= abs(x2 - x1); j++) {
                    image[bmiV3->height - y_c - 1 + ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].r = 255;
                    image[bmiV3->height - y_c - 1 + ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].g = 255;
                    image[bmiV3->height - y_c - 1 + ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].b = 255;
                    image[bmiV3->height - y_c - 1 - ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].r = 255;
                    image[bmiV3->height - y_c - 1 - ((int)
(abs(x2 - x1) / 2) + i)][x1 + j].g = 255;
                }
            }
        }
    }
}

```



```

        image[y2 + i][x2 - j].r = tmp->r;
        image[y2 + i][x2 - j].g = tmp->g;
        image[y2 + i][x2 - j].b = tmp->b;
    }
}

if (abs(y2 - y1) % 2 == 0) {
    for (int i = 0; i < (abs(x2 - x1) + 1) / 2; i++) {
        tmp->r = image[y1 - abs(y2 - y1) / 2][x1 + i].r;
        tmp->g = image[y1 - abs(y2 - y1) / 2][x1 + i].g;
        tmp->b = image[y1 - abs(y2 - y1) / 2][x1 + i].b;
        image[y1 - abs(y2 - y1) / 2][x1 + i].r = image[y1
- abs(y2 - y1) / 2][x2 - i].r;
        image[y1 - abs(y2 - y1) / 2][x1 + i].g = image[y1
- abs(y2 - y1) / 2][x2 - i].g;
        image[y1 - abs(y2 - y1) / 2][x1 + i].b = image[y1
- abs(y2 - y1) / 2][x2 - i].b;
        image[y1 - abs(y2 - y1) / 2][x2 - i].r = tmp->r;
        image[y1 - abs(y2 - y1) / 2][x2 - i].g = tmp->g;
        image[y1 - abs(y2 - y1) / 2][x2 - i].b = tmp->b;
    }
}
free(tmp);
}
}

return 0;
}
}

```

### **TurnImage.h**

```

#ifndef TURNIMAGE_H
#define TURNIMAGE_H

#include "structures.h"
#include "libraries.h"
#include "WriteImage.h"

int TurnImage(BITMAPFILEHEADER* bmfh, BITMAPINFOV3* bmiV3, RGB** image,
int x1, int y1, int x2, int y2, int alpha, char* outputfile);

#endif

```

**WriteImage.c**

```
#include "WriteImage.h"

void WriteImage(BITMAPFILEHEADER* bmfh, BITMAPINFOV3* bmiV3, RGB** image,
char* outputfile){

    FILE *file_write = fopen(outputfile, "wb");
    fwrite(bmfh, sizeof(BITMAPFILEHEADER), 1, file_write);
    fwrite(bmiV3, sizeof(BITMAPINFOV3), 1, file_write);
    for (int i = 0; i < bmiV3->height; i++){
        fwrite(image[i], sizeof(RGB), bmiV3->width, file_write);
        for (int j = 0; j < bmiV3->width % 4; j++){
            fputc(0, file_write);
        }
    }
    fclose(file_write);
}
```

**WriteImage.h**

```
#ifndef WRITEIMAGE_H
#define WRITEIMAGE_H

#include "structures.h"
#include "libraries.h"
void WriteImage(BITMAPFILEHEADER* bmfh, BITMAPINFOV3* bmiV3, RGB** image,
char* outputfile);

#endif
```

**Makefile**

```
all: cw2
```

```
cw2: component.o main.o CircleFromSquare.o CircleFromCenter.o Square.o
TurnImage.o WriteImage.o Swap.o ReadImage.o FreeMemory.o help.o
information.o

        gcc component.o main.o CircleFromSquare.o CircleFromCenter.o
Square.o TurnImage.o WriteImage.o Swap.o ReadImage.o FreeMemory.o help.o
information.o -o cw2
```

```
main.o: main.c
        gcc -c main.c

component.o: component.c component.h
        gcc -c component.c

CircleFromSquare.o: CircleFromSquare.c CircleFromSquare.h
        gcc -c CircleFromSquare.c

CircleFromCenter.o: CircleFromCenter.c CircleFromCenter.h
        gcc -c CircleFromCenter.c

Square.o: Square.c Square.h
        gcc -c Square.c

TurnImage.o: TurnImage.c TurnImage.h
        gcc -c TurnImage.c

WriteImage.o: WriteImage.c WriteImage.h
        gcc -c WriteImage.c

Swap.o: Swap.c Swap.h
        gcc -c Swap.c

ReadImage.o: ReadImage.c ReadImage.h
        gcc -c ReadImage.c

FreeMemory.o: FreeMemory.c FreeMemory.h
        gcc -c FreeMemory.c

help.o: help.c help.h
        gcc -c help.c

information.o: information.c information.h
        gcc -c information.c

clear:
        rm *.o
        rm cw2
```