

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

**ОТЧЕТ**  
**по курсовой работе**  
**по дисциплине «Программирование»**  
**Тема: Обработка изображений на языке Си в формате BMP.**

Студентка гр. 0382

Кривенцова Л.С.

Преподаватель

Берленко Т.А.

Санкт-Петербург

2021

Студентка Кривенцова Л.С.

Группа 0382

Тема работы: Обработка изображений на языке Си в формате BMP.

Исходные данные:

Программа принимает на вход аргументы и изображение в формате BMP. Необходимо преобразовать картинку в соответствии с условиями и сохранить измененную копию. Поддержка операций ведётся через терминальный интерфейс (CLI - Command Line Interface).

Предполагаемый объём пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 16.05.2021

Дата сдачи реферата: 23.05.2021

Дата защиты реферата: 25.05.2021

Студентка гр. 0382

Преподаватель

Кривенцова Л.С.

Берленко Т.А.

## **АННОТАЦИЯ**

В курсовой работе была написана программа, которая реализует обработку изображения в формате BMP в соответствии с заданными условиями. Для управления поддерживается терминальный интерфейс с помощью библиотеки `getopt.h`. В программе представлены следующие опции: рисования квадрата с диагоналями, фильтр rgb-компонент, поворот изображения (части) на 90/180/280 градусов и рисование окружности.

## **SUMMARY**

In the course work, a program was written that implements image processing in BMP format in accordance with the specified conditions. For control, a terminal interface is supported using the `getopt.h` library. The program provides the following options: drawing a square with diagonals, filtering the rgb components, rotating the image (part) by 90/180/280 degrees, and drawing a circle.

## СОДЕРЖАНИЕ

Введение.	...5
1. Цель работы.	...6
2. Задание	...6
Выполнение работы.	...9
1.Ход решения.	...9
2. Функции.	...9
Тестирование.	...13
Заключение.	...16
Список источников.	...17
Приложение А. Исходный код программы.	...18

## **ВВЕДЕНИЕ**

Необходимо написать программу, реализовывающую обработку BMP изображения, выбранную пользователем.

Программа обеспечивает работу алгоритма на операционной системе на базе Linux. Написание производилось с помощью IDE Clion на операционной системе Ubuntu.

Результатом работы является программа, производящая считывание, затем обработку и сохранение изображения в формате BMP в бинарном виде. Программой используется динамическая память, которая очищается перед завершением. Программа также обрабатывает все случаи некорректного обращения, выводя сообщение о соответствующей ошибке.

## **Цель работы.**

Изучить основы обработки изображения в формате BMP и написать программу, производящую эти действия на языке Си.

## **Задачи.**

Обеспечить бинарное считывание изображения с использованием соответствующих структур;

Реализовать вспомогательные функции, производящие обработку бинарных данных (изображения);

Создать и сохранить копию исходного изображения с примененными изменениями;

Реализовать терминальный интерфейс CLI.

## **Задание.**

Вариант 9

Программа должна иметь CLI или GUI.

## **Общие сведения**

- 24 бита на цвет
- без сжатия
- файл всегда соответствует формату BMP (но стоит помнить, что версий у формата несколько)
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей

- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

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 какое 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



## **ВЫПОЛНЕНИЕ РАБОТЫ.**

### **Ход решения:**

Используются заголовочные файлы стандартной библиотеки языка Си `stdio.h`, `string.h` (для работы со строками), `stdlib.h`, `unistd.h`, `stdint.h` (для работы со структурами бинарного представления изображения) и `getopt.h` (для работы с терминальным интерфейсом).

На вход программы подаются ключи и их аргументы (различные для разных подзадач, но по умолчанию последним аргументом передаётся имя выходного файла). Происходит считывание изображения в специальные структуры, отвечающие за хранение информации (заголовка) изображения. Динамически выделяется память для массива структур RGB (структура состоит из трёх полей, отвечающих за цвета пикселя) — этот двумерный массив и представляет собой картинку. При считывании учитывается выравнивание изображения, которое вычисляется по формуле «4 — ([длина изображения]\*3)%4».

Далее происходит обработка различных случаев (с помощью `switch`) полученных программой ключей — в переменные сохраняются аргументы, полученные программой с помощью `getopt_long`. Для работы с `getopt_long` предназначены строка `optString`, хранящая допустимые ключи и, в качестве константы, структура `option longOpts[]`, описывающая длинные и сокращённые ключи. В зависимости от полученного ключа вызывается соответствующая функция по обработке изображения. Далее происходит очищение динамической памяти.

### **Функции.**

#### **1. Главная.**

Функция осуществляет открытие файла в переменную `filein` с помощью `fopen`, затем записывает в `bmfh` и `bmih` (объекты структур `BITMAPINFOHEADER` и `BITMAPFILEHEADER`) информацию из заголовка

файла. Выделяется динамическая память для массива пикселей `arg` и массива, отвечающего за выравнивание `pad`. Далее в массив пикселей записывается исходное изображение и файл закрывается с помощью `fclose`. В переменную `opt` считывается `getopt_long`. Если переменная равна -1 (то есть ключи не были переданы), вызывается функция `printMenu`, печатающая меню программы. Далее через случаи `switch` обрабатываются переданные ключи, затем происходит очищение динамической памяти.

## **2. *RGB Colorize(unsigned char r, unsigned char g, unsigned char b).***

Функция отвечает за раскраску пикселя (ячейки двумерного массива), присваивая его полям `r,g,b`, отвечающим за цвета, значения, переданные функции `arg` в качестве аргументов. Функция возвращает структуру `RGB` - раскрашенный пиксель.

## **3. *void imageInfo(BITMAPFILEHEADER bfh, BITMAPINFOHEADER bih).***

Функция принимает на вход два аргумента — объекты структур `BITMAPFILEHEADER` и `BITMAPINFOHEADER` и, ничего не возвращая, печатает информацию об изображении, хранящуюся в данных структурах.

## **4. *int imageOkay()***

Функция не получает на вход аргументов. Она проверяет, соответствует ли формат изображения обрабатываемым случаям с помощью полей структуры `BITMAPINFOHEADER` — `biCompression` и `biClrUsed`. Если изображение имеет приемлимый формат, функция возвращает 0, иначе значение 1.

## **5. *void outputF()*.**

Функция не принимает на вход аргументы и ничего не возвращает. С помощью `for` она открывает файл в переменную `fileout`, из объектов структур `BITMAPFILEHEADER` и `BITMAPINFOHEADER` записывает в файл заголовок информации о новом изображении, затем во вложенных

циклах `for` в файл записывается само изображение в бинарном виде (из массива пикселей). Далее файл закрывается с помощью `fclose`.

#### **6. `void drawSquare(int ctrX, int ctrY, int st, int thick)`.**

Функция реализует рисование квадрата, левый верхний угол которого находится в координатах (**`ctrX`**, **`ctrY`**), которые функция принимает на вход в качестве аргументов, как и значения стороны квадрата и толщины линий. Вызывается функция `imageOkay`, и если изображение читаемо, то проходит проверка, корректно ли указаны координаты: если нет, то функция выводит соответственное сообщение и завершается. Затем, если квадрат должен быть нарисован с заливкой (о чём сообщает переменная-флаг `fill`), во вложенных циклах `for`, проходя по координатам, закрашивается квадрат. В след за этим, в аналогичных циклах рисуются стороны и диагонали квадрата. В итоге вызывается функция `outputF`, сохраняя получившееся изображение. Функция ничего не возвращает.

#### **7. `void componentRBG(char* color, int lot)`.**

Функция принимает на вход два аргумента: строку, отвечающую за цвет и целочисленную переменную и устанавливает указанный цвет в переданное число, проходясь по всему массиву пикселей во вложенных циклах `for`.

#### **8. `void drawCircle(int x0, int y0, int rad, int thick)`.**

Функция реализует рисование круга, центр которого находится в координатах (`x0`, `y0`), которые функция принимает на вход в качестве аргументов, как и значения радиуса и толщины линий. Вызывается функция `imageOkay`, и если изображение читаемо, то проходит проверка, корректно ли указаны координаты: если нет, то функция выводит соответственное сообщение и завершается. В след за этим, рисуется окружность и регулируется толщина её контура. Затем, если круг должен быть нарисован с заливкой (о чём сообщает переменная-флаг `fill`), во вложенных циклах `for`, проходя по координатам, закрашивается внутренность круга. В итоге

вызывается функция `outputF`, сохраняя получившееся изображение. Функция ничего не возвращает.

#### **9. `void turn( int x0, int y0,int x_e, int y_e, int corner)`.**

Функция реализует поворот области, заданной координатами, переданными функции в качестве аргументов, как и значение угла поворота. Вызывается функция `imageOkay`, и если изображение читаемо, то проходит проверка, корректно ли указаны координаты: если нет, то функция выводит соответственное сообщение и завершается. Если координаты описывают изображение целиком, то вызывается функция `rotate`, как отдельный случай (поворот всей картинке) и функция завершается, а если требуется поворот только части изображения, то функция продолжается: высчитываются вспомогательные координаты и матрица поворота (с помощью `switch` определяются случаи углов, если угол указан неверно, то функция выводит соответственное сообщение и завершается). Заданная область окрашивается в белый. Создаётся копия картинке — двумерный массив `sourarr`, из которого затем и копируются пиксели при повороте. Во вложенных циклах `for` реализуется поворот части картинке, затем очищается память вспомогательного массива `sourarr`. Функция ничего не возвращает.

#### **10. `void rotate(int corner)`.**

Функция принимает на вход угол поворота картинке целиком, в качестве целочисленного аргумента. Делит по случаям с помощью `switchЖ` если требуется повернуть картинку на 90 или 180 градусов, то поля ширины и высоты картинке меняются друг с другом, заново происходит расчёт выравнивания изображения. Затем в массив `arr1` происходит копирование пикселей из основного массива `arr` с пересчётом координат: за счёт этого и происходит поворот картинке. Вызывается функция `outputF()` для сохранения получившегося изображения. В случае, когда угол равен 270 градусам происходит всё то же самое, помимо смены местами значений полей ширины и высоты, там они остаются неизменными. Если угол указан неверно, то

функция выводит соответственное сообщение и завершается. Затем происходит освобождение памяти динамического массива `arr1`.

#### **11. *void printMenu()*.**

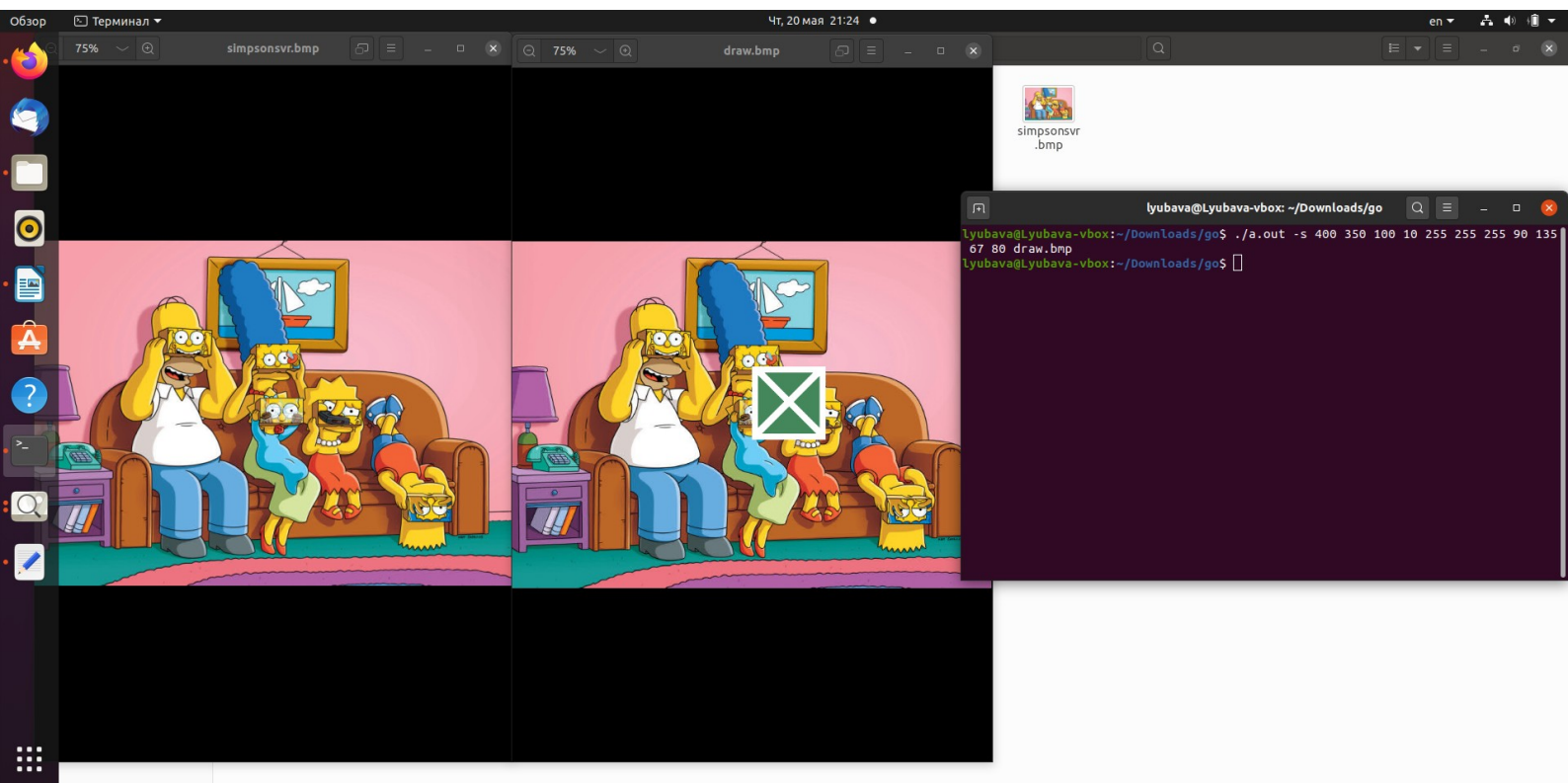
Функция не получает на вход аргументы и ничего не возвращает. Печатает меню программы — список доступных опций и ключи, по которым к ним можно обратиться.

Разработанный программный код см. в приложении А.

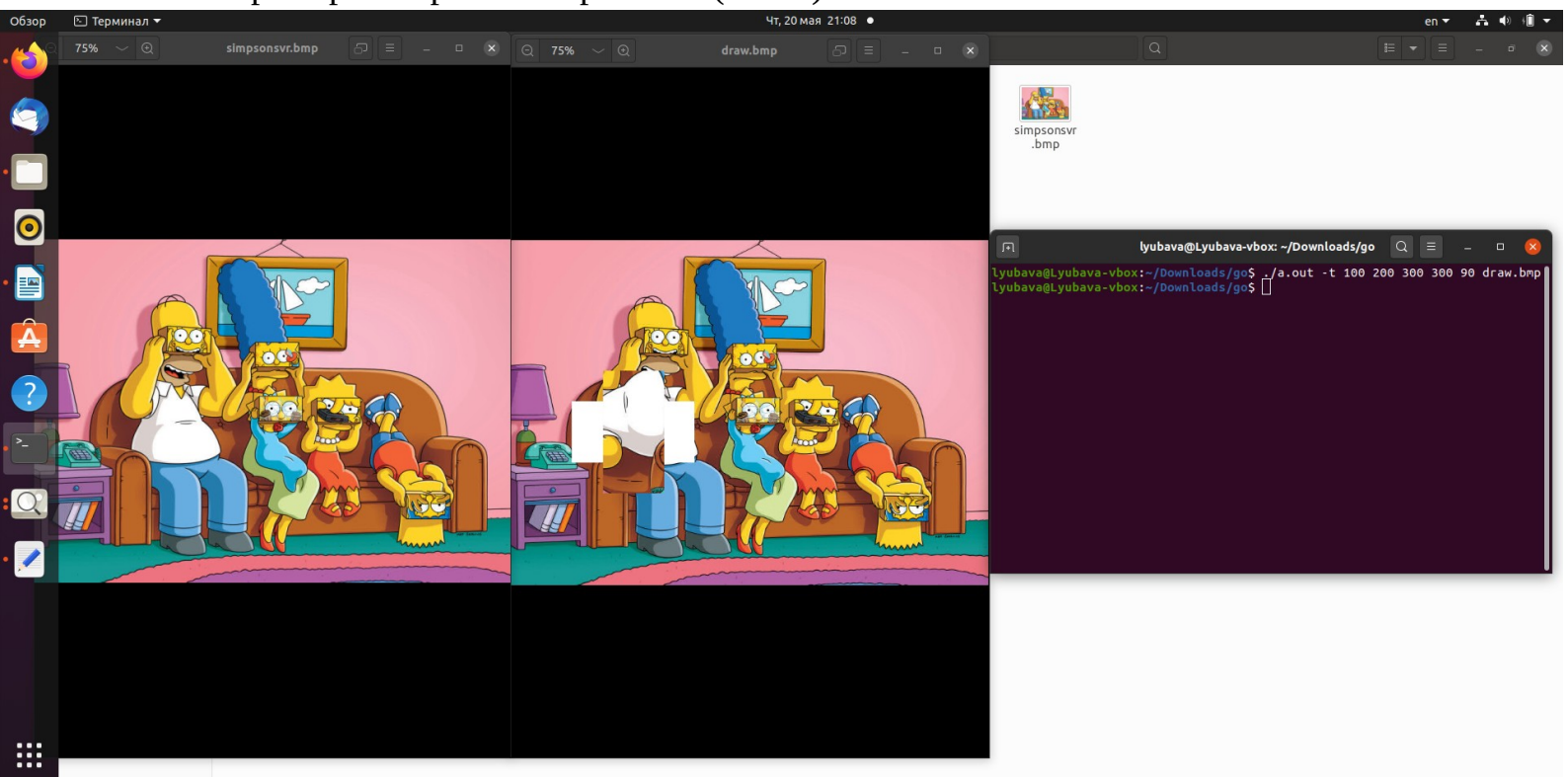
## ТЕСТИРОВАНИЕ

Результаты тестирования.

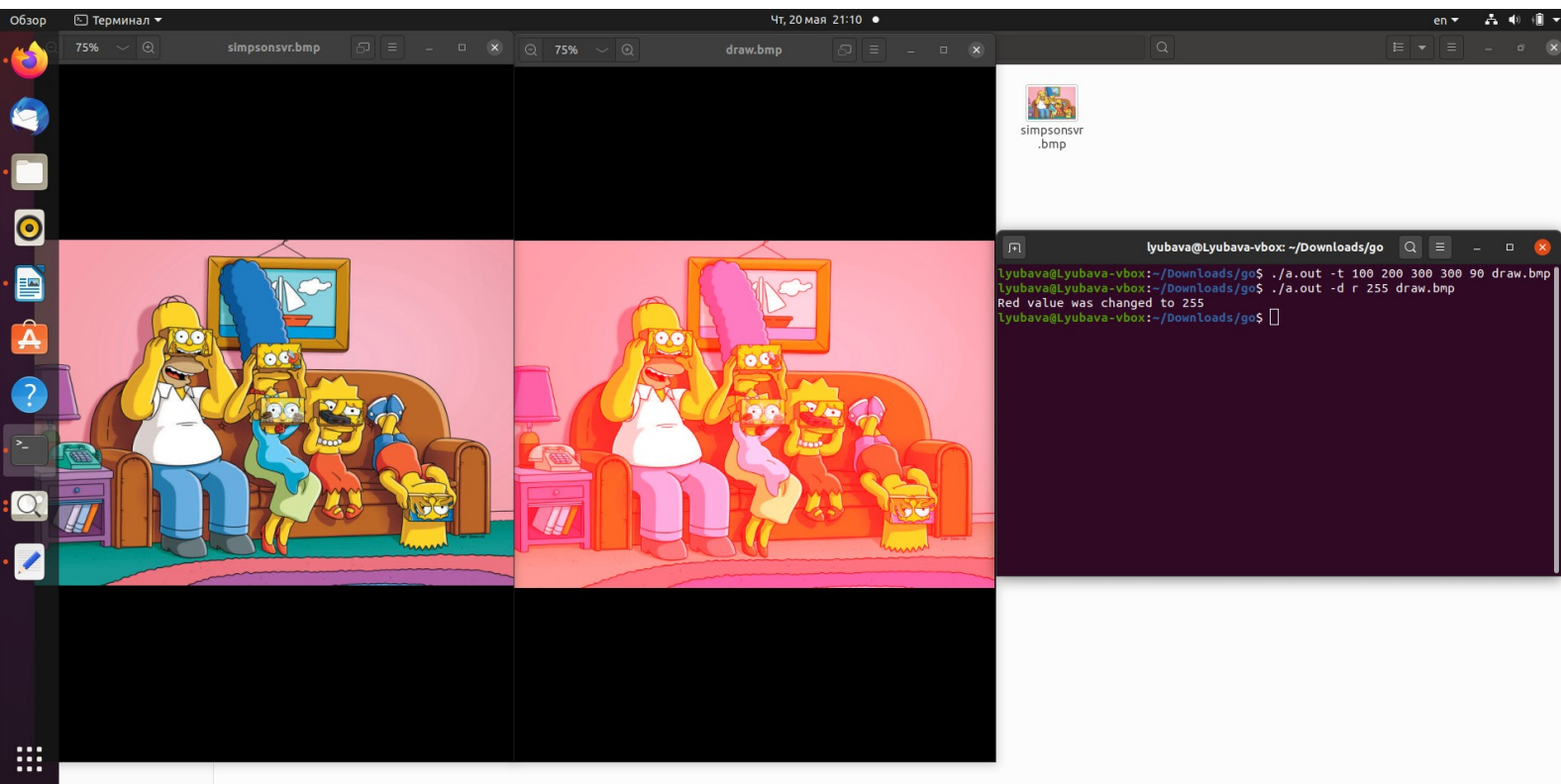
Пример рисования квадрата.



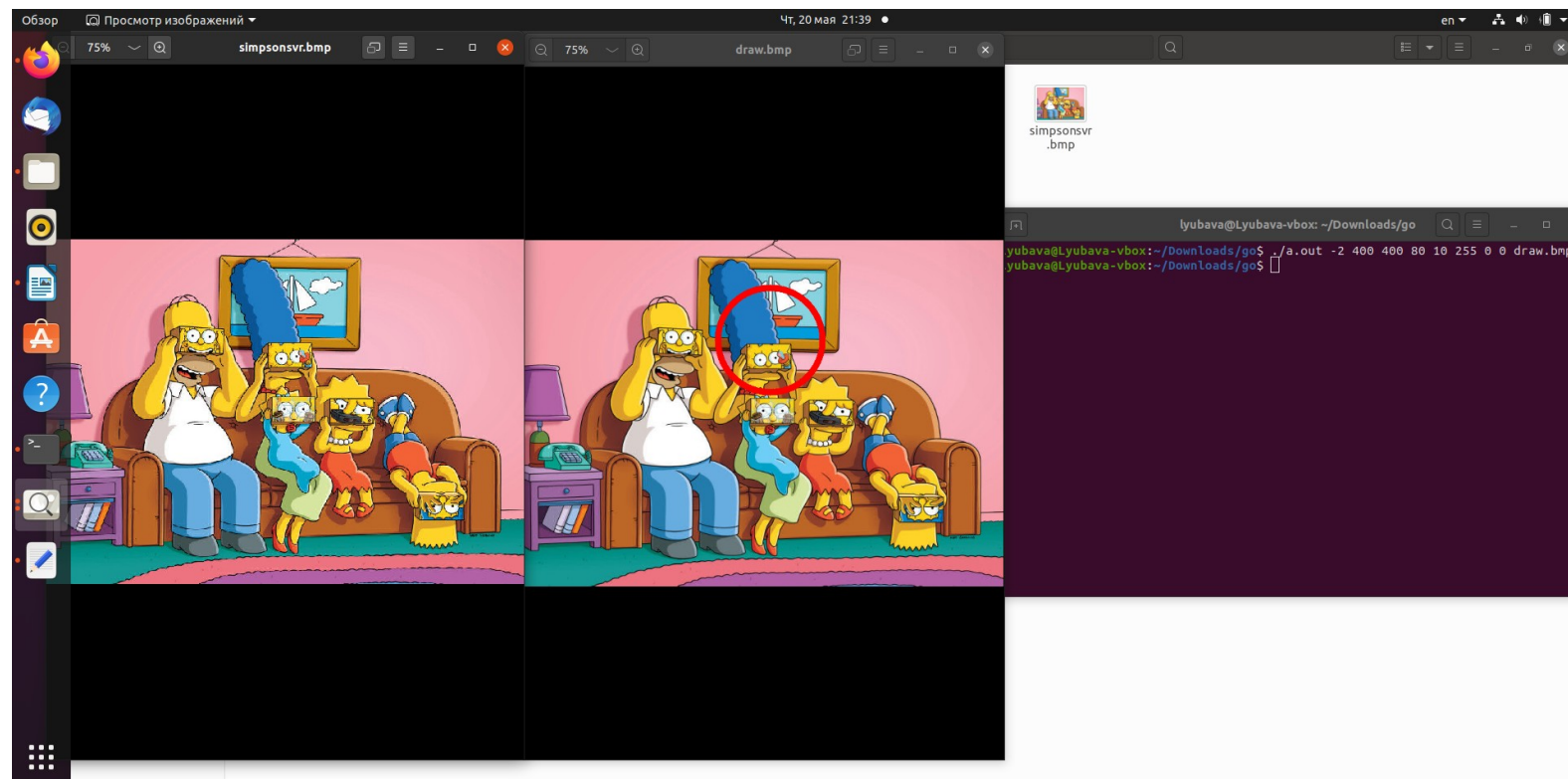
Пример поворота изображения(части).



Пример работы фильтра rgb-компонент.

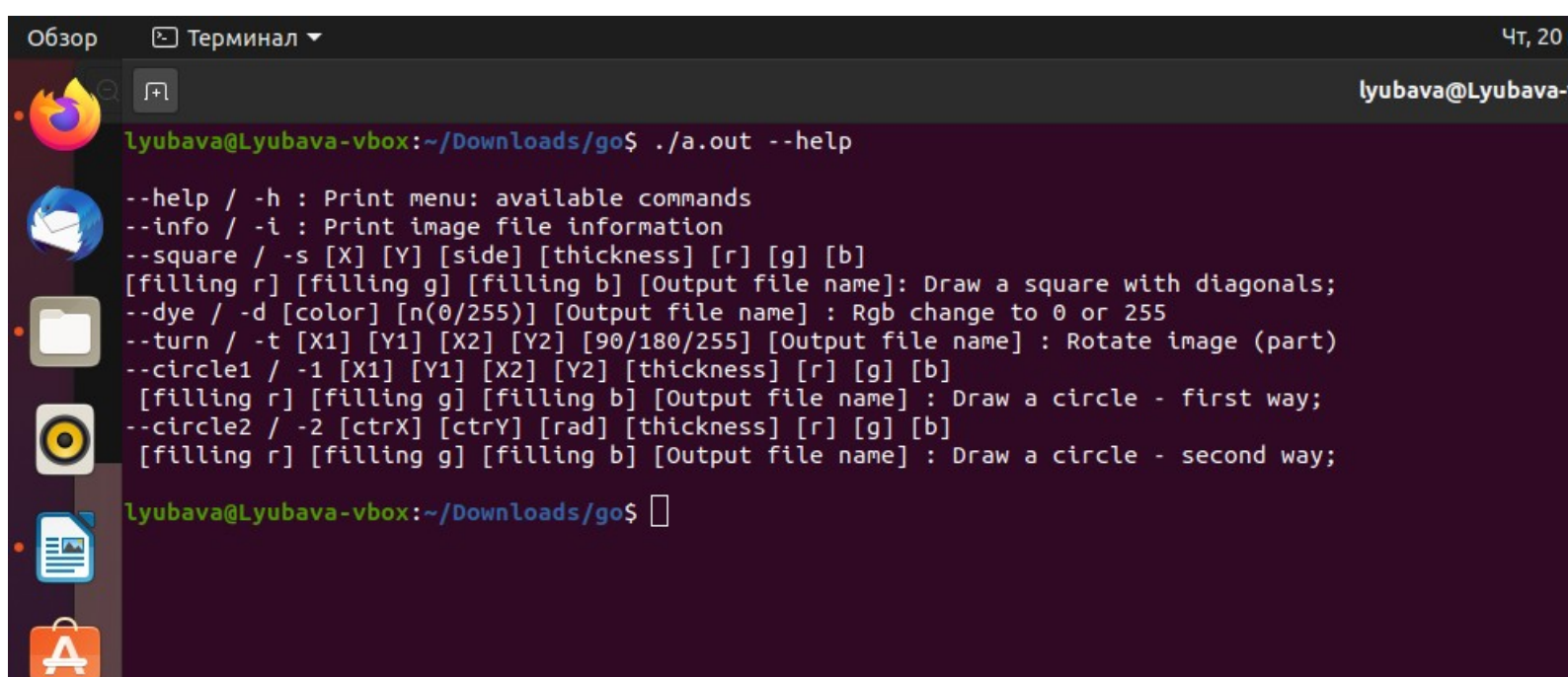


Пример рисования окружности.





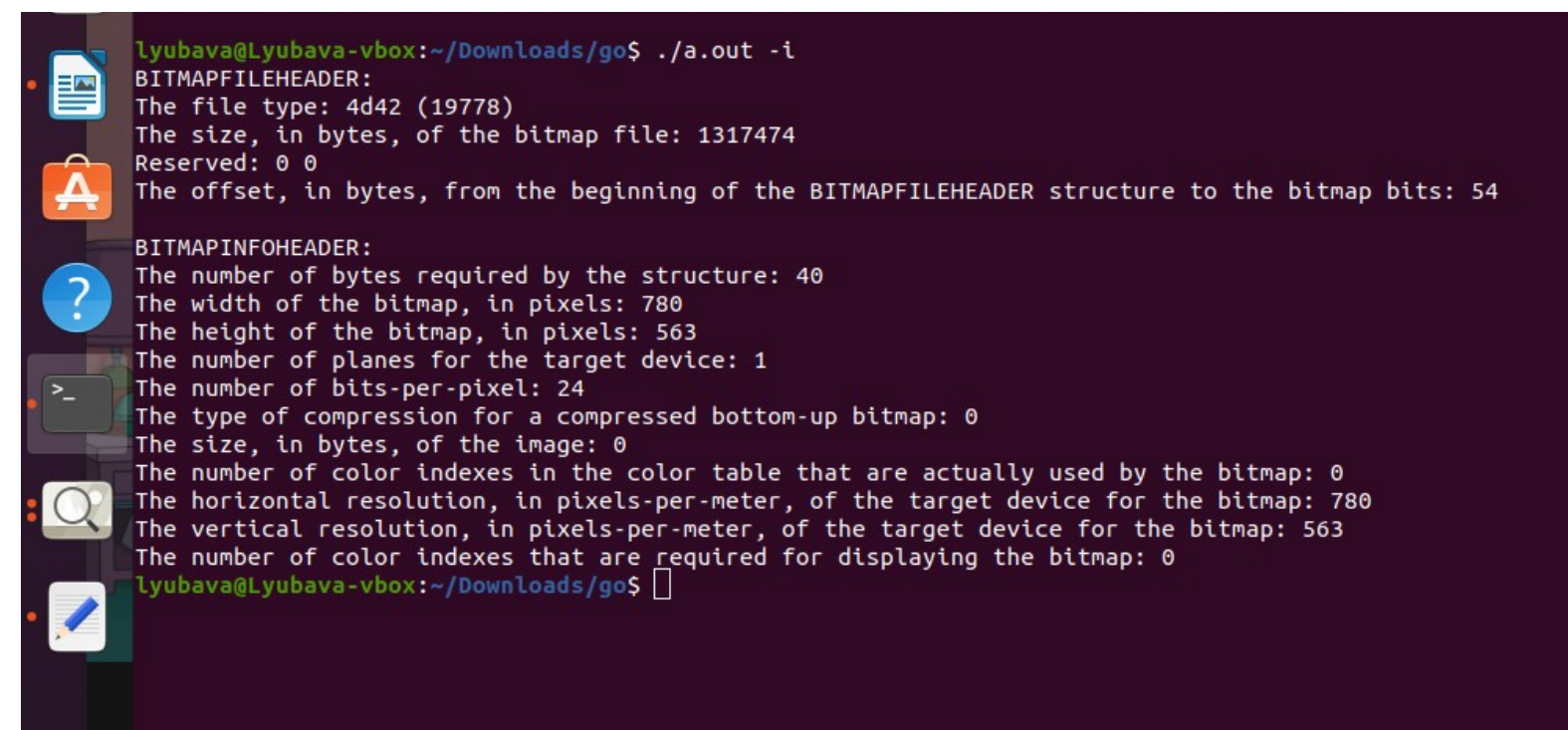
Пример вывода меню.



Обзор Терминал Чт, 20 lyubava@Lyubava-

```
lyubava@Lyubava-vbox:~/Downloads/go$ ./a.out --help
--help / -h : Print menu: available commands
--info / -i : Print image file information
--square / -s [X] [Y] [side] [thickness] [r] [g] [b]
[filling r] [filling g] [filling b] [Output file name]: Draw a square with diagonals;
--dye / -d [color] [n(0/255)] [Output file name] : Rgb change to 0 or 255
--turn / -t [X1] [Y1] [X2] [Y2] [90/180/255] [Output file name] : Rotate image (part)
--circle1 / -1 [X1] [Y1] [X2] [Y2] [thickness] [r] [g] [b]
[filling r] [filling g] [filling b] [Output file name] : Draw a circle - first way;
--circle2 / -2 [ctrX] [ctrY] [rad] [thickness] [r] [g] [b]
[filling r] [filling g] [filling b] [Output file name] : Draw a circle - second way;
lyubava@Lyubava-vbox:~/Downloads/go$
```

Пример вывода информации об изображении.



```
lyubava@Lyubava-vbox:~/Downloads/go$ ./a.out -i
BITMAPFILEHEADER:
The file type: 4d42 (19778)
The size, in bytes, of the bitmap file: 1317474
Reserved: 0 0
The offset, in bytes, from the beginning of the BITMAPFILEHEADER structure to the bitmap bits: 54

BITMAPINFOHEADER:
The number of bytes required by the structure: 40
The width of the bitmap, in pixels: 780
The height of the bitmap, in pixels: 563
The number of planes for the target device: 1
The number of bits-per-pixel: 24
The type of compression for a compressed bottom-up bitmap: 0
The size, in bytes, of the image: 0
The number of color indexes in the color table that are actually used by the bitmap: 0
The horizontal resolution, in pixels-per-meter, of the target device for the bitmap: 780
The vertical resolution, in pixels-per-meter, of the target device for the bitmap: 563
The number of color indexes that are required for displaying the bitmap: 0
lyubava@Lyubava-vbox:~/Downloads/go$
```



## **ЗАКЛЮЧЕНИЕ**

Были изучены основы обработки BMP изображения на языке Си.

Разработана программа, производящая считывание, затем обработку и сохранение изображения в формате BMP в бинарном виде. Программой используется динамическая память, которая очищается перед завершением. Программа также обрабатывает все случаи некорректного обращения, выводя сообщение о соответствующей ошибке. Взаимодействие с пользователем происходит с помощью CLI.

## СПИСОК ИСТОЧНИКОВ

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

<https://www.gnu.org/software/libc/manual/pdf/libc.pdf>

[http://www.vsokovikov.narod.ru/New\\_MSDN\\_API/Bitmaps/  
str\\_bitmapinfoheader.htm](http://www.vsokovikov.narod.ru/New_MSDN_API/Bitmaps/str_bitmapinfoheader.htm)

<https://ru.wikipedia.org/wiki/%D0%9C>

%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0\_%D0%BF%D0%BE  
%D0%B2%D0%BE%D1%80%D0%BE%D1%82%D0%B0

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: cw.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <getopt.h>

#pragma pack(push, 1)
typedef struct{
    unsigned int biSize;
    unsigned int biWidth;
    unsigned int biHeight;
    uint16_t biPlanes;
    uint16_t biBitCount;
    unsigned int biCompression;
    unsigned int biSizeImage;
    int biXPelsPerMeter;
    int biYPelsPerMeter;
    unsigned int biClrUsed;
    unsigned int biClrImportant;
}BITMAPINFOHEADER;

typedef struct{
    uint16_t bfType;
    unsigned int bfSize;
    uint16_t bfReserved1;
    uint16_t bfReserved2;
    unsigned int bfOffBits;
}BITMAPFILEHEADER;

typedef struct{
    unsigned char red;
    unsigned char green;
    unsigned char blue;
}RGB;

#pragma pack(pop)

static const struct option longOpts[] = {
    { "info", no_argument, NULL, 'i' },
    { "square", required_argument, NULL, 's' },
    { "circle1", required_argument, NULL, '1' },
    { "circle2", required_argument, NULL, '2' },
    { "turn", required_argument, NULL, 't' },
    { "dye", required_argument, NULL, 'd' },
    { "help", no_argument, NULL, 'h' },
    { NULL, no_argument, NULL, 0 }
};

static const char *optString = "is:2:r:d:t:1:ho?";
```

```

RGB **arr, **copyarr, **arr1, *pad;
BITMAPFILEHEADER bmfh;
BITMAPINFOHEADER bmih;
FILE *filein, *fileout;
size_t padding;
unsigned char r = 0, g = 0, b = 0;
unsigned char fr = 0, fg = 0, fb = 0;
char *inputName, *outputName = "draw.bmp";
char whichColor[5];
int colorCount, fill=0, flag =0;

RGB Colorize(unsigned char r, unsigned char g, unsigned char b){
    RGB pixel;
    pixel.red = r;
    pixel.green = g;
    pixel.blue = b;
    return pixel;
}

void imageInfo(BITMAPFILEHEADER bfh, BITMAPINFOHEADER bih){
    printf("BITMAPFILEHEADER:\n");
    printf("The file type: %x (%hu)\n", bfh.bfType, bfh.bfType);
    printf("The size, in bytes, of the bitmap file: %i\n",
bfh.bfSize);
    printf("Reserved: %i %i\n", bfh.bfReserved1, bfh.bfReserved2);
    printf("The offset, in bytes, from the beginning of the
BITMAPFILEHEADER structure to the bitmap bits: %i\n\n",
bfh.bfOffBits);
    printf("BITMAPINFOHEADER:\n");
    printf("The number of bytes required by the structure: %i\n",
bih.biSize);
    printf("The width of the bitmap, in pixels: %i\n", bih.biWidth);
    printf("The height of the bitmap, in pixels: %i\n", bih.biHeight);
    printf("The number of planes for the target device: %i\n",
bih.biPlanes);
    printf("The number of bits-per-pixel: %i\n", bih.biBitCount);
    printf("The type of compression for a compressed bottom-up bitmap:
%i\n", bih.biCompression);
    printf("The size, in bytes, of the image: %i\n", bih.biSizeImage);
    printf("The number of color indexes in the color table that are
actually used by the bitmap: %i\n", bih.biClrUsed);
    printf("The horizontal resolution, in pixels-per-meter, of the
target device for the bitmap: %i\n", bih.biWidth);
    printf("The vertical resolution, in pixels-per-meter, of the
target device for the bitmap: %i\n", bih.biHeight);
    printf("The number of color indexes that are required for
displaying the bitmap: %i\n", bih.biClrImportant);
}

int imageOkay(){
    if(bmih.biCompression != 0){
        printf("Unknown compression method\n");
        return 1;
    }
    if(bmih.biClrUsed != 0){

```

```

        printf("Version not supported\n");
        return 1;
    }
    return 0;
}

void outputF(){
    fileout = fopen(outputName, "w+b");
    fwrite(&bmfh, sizeof(bmfh), 1, fileout);
    fwrite(&bmih, sizeof(bmih), 1, fileout);

    for(int i = 0; i < bmih.biHeight; i++){
        for(int j = 0; j < bmih.biWidth; j++){
            if (flag == 1)
                fwrite(&arr1[i][j]), sizeof(RGB), 1, fileout);
            else fwrite(&arr[i][j]), sizeof(RGB), 1, fileout);
            if(padding != 0)
                fwrite(pad, padding, 1, fileout);
        }
    }
    fclose(fileout);
}

void drawSquare(int ctrX, int ctrY, int st, int thick){
    if(imageOkay())
        return;
    if(ctrX < 0 || ctrY < 0 || ctrX > bmih.biWidth || ctrX + st >
bmih.biWidth || ctrY > bmih.biHeight || ctrY + st > bmih.biHeight ||
ctrY + st < 0 || ctrX + st < 0){
        printf("Impossible coordinates\n");
        return;
    }
    int x,y;
    if (fill==1)
        for( y = 0; y<st;y++){
            for( x = 0; x<st;x++){
                if (ctrY-y >= 0 && ctrY-y<= bmih.biHeight && ctrX+x>=0 &&
ctrX+x <= bmih.biWidth)
                    arr[ctrY-y][ctrX+x]= Colorize(fr,fg,fb);
            }
        }
    for(y=-thick+1; y<= 0; y++){
        for( x = -thick; x<st+thick;x++){
            if (ctrY-y >= 0 && ctrY-y<= bmih.biHeight && ctrX+x>=0 &&
ctrX+x <= bmih.biWidth)
                arr[ctrY-y][ctrX+x]= Colorize(r,g,b);
            if (ctrY-st+y >= 0 && ctrY-st+y<= bmih.biHeight && ctrX+x>=0
&& ctrX+x <= bmih.biWidth)
                arr[ctrY-st+y][ctrX+x]= Colorize(r,g,b);
        }
    }
    for( x =-thick+1; x<=0;x++){
        for( y = 0; y<st;y++){

```

```

        if (ctrY-y >= 0 && ctrY-y<= bmih.biHeight && ctrX+x-1>=0 &&
ctrX+x-1 <= bmih.biWidth)
            arr[ctrY-y][ctrX+x-1]= Colorize(r,g,b);
        if (ctrY-y >= 0 && ctrY-y<= bmih.biHeight && ctrX+st-x>=0 &&
ctrX+st-x <= bmih.biWidth)
            arr[ctrY-y][ctrX+st-x]= Colorize(r,g,b);
    }
}
for(y=0; y< st; y++){
    for( x = y-(thick/2); x<y+thick;x++){
        if (ctrY-y >= 0 && ctrY-y<= bmih.biHeight && ctrX+x>=0 &&
ctrX+x <= bmih.biWidth)
            arr[ctrY-y][ctrX+x]= Colorize(r,g,b);
    }
}
for(y=0; y< st; y++){
    for( x = st-y+(thick/2); x>=st-y-thick;x--){
        if (ctrY-y >= 0 && ctrY-y<= bmih.biHeight && ctrX+x>=0 &&
ctrX+x <= bmih.biWidth)
            arr[ctrY-y][ctrX+x]= Colorize(r,g,b);
    }
}
outputF();
}

void componentRBG(char* color, int lot){
    if(imageOkay())
        return;
    if(strcmp(color, "r") == 0){
        for(int i = 0; i < bmih.biHeight; i++)
            for(int j = 0; j < bmih.biWidth; j++)
                arr[i][j].blue = (unsigned char)lot;
        printf("Red value was changed to %u\n", (unsigned
char)lot);
    }
    if(strcmp(color, "g") == 0){
        for(int i = 0; i < bmih.biHeight; i++)
            for(int j = 0; j < bmih.biWidth; j++)
                arr[i][j].green = (unsigned char)lot;
        printf("Green value was changed to %u\n", (unsigned
char)lot);
    }
    if(strcmp(color, "b") == 0){
        for(int i = 0; i < bmih.biHeight; i++)
            for(int j = 0; j < bmih.biWidth; j++)
                arr[i][j].red = (unsigned char)lot;
        printf("Blue value was changed to %u\n", (unsigned
char)lot);
    }
    outputF();
}

void drawCircle(int x0, int y0, int rad, int thick){
    if(imageOkay())
        return;

```

```

        if(x0 < 0 || x0 - rad < 0 || y0 < 0 || y0 - rad < 0 || x0 >
bmih.biWidth || x0 + rad > bmih.biWidth || y0 > bmih.biHeight || y0 +
rad > bmih.biHeight || rad<=0){
    printf("Impossible coordinates\n");
    return;
}
int x = 0;
int y = rad;
int delta = 1 - 2 * rad;
int error = 0;
int dd;
while(y >= 0){
    arr[y0 + y][x0 + x] = Colorize(r, g, b);
    arr[y0 - y][x0 + x] = Colorize(r, g, b);
    arr[y0 + y][x0 - x] = Colorize(r, g, b);
    arr[y0 - y][x0 - x] = Colorize(r, g, b);
    error = 2 * (delta + y) - 1;
    if((delta < 0) && (error <= 0)){
        delta += 2 * ++x + 1;
        continue;
    }
    if((delta > 0) && (error > 0)){
        delta -= 2 * (--y) + 1;
        continue;
    }
    delta += 2 * (++x - y--);
}
for(int i = 0; i < bmih.biHeight; i++)
    for(int j = 0; j < bmih.biWidth; j++)
        if((j-x0)*(j-x0)+(i-x0)*(i-x0)>=rad*rad &&
(j-x0)*(j-x0)+(i-y0)*(i-y0)<=(rad+thick)*(rad+thick))
arr[i][j] = Colorize(r, g, b);
if (fill==1)
    for(int i = 0; i < bmih.biHeight; i++)
        for(int j = 0; j < bmih.biWidth; j++)
            if((j-x0)*(j-x0)+(i-y0)*(i-y0)<=rad*rad) arr[i][j] =
Colorize(fr, fg, fb);
outputF();
}

```

```

void rotate(int corner){
int change;
switch (corner){
    case 90:
        change = bmih.biWidth;
        bmih.biWidth = bmih.biHeight;
        bmih.biHeight = change;
        if((bmih.biWidth * 3) % 4)
            padding = 4 - (bmih.biWidth * 3) % 4;
        arr1 = (RGB**) malloc(bmih.biHeight * sizeof(RGB*));
        for(int i = 0; i < bmih.biHeight; i++){
            arr1[i] = (RGB*)malloc(bmih.biWidth * sizeof(RGB));
            for (int j = 0; j < bmih.biWidth; j++){
                arr1[i][j]=Colorize(255,255,255);
                arr1[i][j]=arr[j][bmih.biHeight-1-i];
            }
        }
    }
}

```

```

        }
    }
    flag = 1;
    outputF();
    break;
case 180:
    change = bmih.biWidth;
    bmih.biWidth = bmih.biHeight;
    bmih.biHeight = change;
    if((bmih.biWidth * 3) % 4)
        padding = 4 - (bmih.biWidth * 3) % 4;
    arr1 = (RGB**) malloc(bmih.biHeight * sizeof(RGB*));
    for(int i = 0; i < bmih.biHeight; i++)
        arr1[i] = (RGB*)malloc(bmih.biWidth * sizeof(RGB));
    for(int i = 0; i < bmih.biHeight; i++)
        for (int j = 0; j < bmih.biWidth; j++)
            arr1[i][j]=Colorize(255,255,255);
    for(int i = 0; i < bmih.biHeight; i++)
        for (int j = 0; j <bmih.biWidth; j++){
            arr1[i][j]=arr[bmih.biWidth-1-j][i];
        }
    flag = 1;
    outputF();
    break;
case 270:

    arr1 = (RGB**) malloc(bmih.biHeight * sizeof(RGB*));
    for(int i = 0; i < bmih.biHeight; i++)
        arr1[i] = (RGB*)malloc(bmih.biWidth * sizeof(RGB));
    for(int i = 0; i < bmih.biHeight; i++)
        for (int j = 0; j < bmih.biWidth; j++)
            arr1[i][j]=arr[i][j];
    for(int i = 0; i < bmih.biHeight; i++)
        for (int j = 0; j <bmih.biWidth; j++){
            arr[i][j]=arr1[bmih.biHeight-1-i][bmih.biWidth-1-
j]];
        }
    outputF();
    break;
default:
    printf("You wrote incorrect corner\n");
    return;
    for(int i = 0; i < bmih.biHeight; i++)
        free(arr1[i]);
    free(arr1);
}

}

void turn( int x0, int y0,int x_e, int y_e, int corner){
if(imageOkay())
    return;
int new_x, new_y;
int matrix[2][2];
    if (x0 < 0 || y0 < 0 || x_e < 0 || y_e < 0 || x0 > bmih.biWidth
|| y0 > bmih.biHeight || x_e > bmih.biWidth || y_e > bmih.biHeight){
    printf("Impossible coordinates");

```



```

        return;}
    if ( x0 == 0 && y0 == 0 && x_e == bmih.biWidth && y_e ==
bmih.biHeight){
        rotate(corner);
        return;}

copyarr = (RGB**) malloc(bmih.biHeight * sizeof(RGB*));
for(int i = 0; i < bmih.biHeight; i++)
    copyarr[i] = (RGB*)malloc(bmih.biWidth * sizeof(RGB));
for(int i = 0; i < bmih.biHeight; i++)
    for (int j = 0; j < bmih.biWidth; j++)
        copyarr[i][j]=arr[i][j];

int x_c = x0 + (x_e - x0) / 2;
int y_c = y0 + (y_e - y0) / 2;
for(int y = y0; y < y_e; y++){
    for(int x = x0; x < x_e; x++){
        if(y<bmih.biHeight && x<bmih.biWidth && y>=0 && x>=0)
            arr[y][x]=Colorize(255, 255, 255);
    }
}
switch (corner){
    case 90:
        matrix[0][0] = 0;
        matrix[0][1] = -1;
        matrix[1][0] = 1;
        matrix[1][1] = 0;
        break;
    case 180:
        matrix[0][0] = -1;
        matrix[0][1] = 0;
        matrix[1][0] = 0;
        matrix[1][1] = -1;
        break;
    case 270:
        matrix[0][0] = 0;
        matrix[0][1] = 1;
        matrix[1][0] = -1;
        matrix[1][1] = 0;
        break;
    default:
        printf("You wrote incorrect corner\n");
        return;
}

for(int y = -(y_e - y0)/2; y < (y_e - y0)/2; y++){
    for (int x = -(x_e - x0)/2; x < (x_e - x0)/2; x++) {

        new_y = matrix[1][0]*x + matrix[1][1]*y + y_c;
        new_x = matrix[0][0]*x + matrix[0][1]*y + x_c;
        if(new_y<bmih.biHeight && new_x<bmih.biWidth && new_y>=0
&& new_x>=0)
            arr[new_y][new_x ]=copyarr[y + y_c][(x + x_c) ];
    }
}
for(int i = 0; i < bmih.biHeight; i++)

```

```

        free(copyarr[i]);
        free(copyarr);
        outputF();
    }

void printMenu(){
    printf("\n--help / -h : Print menu: available commands\n");
    printf("--info / -i : Print image file information\n");
    printf("--square / -s [X] [Y] [side] [thickness] [r] [g] [b] \
n[filling r] [filling g] [filling b] [Output file name]: Draw a square
with diagonals;\n");
    printf("--dye / -d [color] [n(0/255)] [Output file name] : Rgb
change to 0 or 255\n");
    printf("--turn / -t [X1] [Y1] [X2] [Y2] [90/180/255] [Output file
name] : Rotate image (part)\n");
    printf("--circle1 / -1 [X1] [Y1] [X2] [Y2] [thickness] [r] [g]
[b]\n [filling r] [filling g] [filling b] [Output file name] : Draw a
circle - first way;\n");
    printf("--circle2 / -2 [ctrX] [ctrY] [rad] [thickness] [r] [g]
[b]\n [filling r] [filling g] [filling b] [Output file name] : Draw a
circle - second way;\n\n");
}

int main(int argc, char *argv[]){
    int x,y,rad,thick;
    int x1,y1, corner = 0;
    inputName = malloc(100 * sizeof(char));
    strcpy(inputName, "simpsonsvr.bmp");
    outputName = malloc(strlen(argv[argc-1]) * sizeof(char));
    strcpy(outputName, argv[argc-1]);
    int opt ;
    filein = fopen(inputName, "r+b");
    fread(&bmfh, sizeof(bmfh), 1, filein);
    fread(&bmiH, sizeof(bmiH), 1, filein);

    arr = (RGB**) malloc(bmiH.biHeight * sizeof(RGB*));
    for(int i = 0; i < bmiH.biHeight; i++)
        arr[i] = (RGB*)malloc(bmiH.biWidth * sizeof(RGB));

    pad = (RGB*)malloc(4 * sizeof(RGB));
    padding = 0;
    if((bmiH.biWidth * 3) % 4)
        padding = 4 - (bmiH.biWidth * 3) % 4;

    for(int i = 0; i < bmiH.biHeight; i++){
        for (int j = 0; j < bmiH.biWidth; j++){
            fread(&(arr[i][j]), sizeof(RGB),1, filein);
            if(padding != 0)
                fread(pad, padding, 1, filein);
        }
    }
    fclose(filein);
    int longIndex;
    int i=1;
    opt = getopt_long( argc, argv, optString, longOpts, &longIndex);
    if(opt == -1)

```

```

        printMenu();
        while(opt != -1)
        {
            switch(opt){
                case 'd':
                    if (argc>2) strcpy(whichColor,argv[2] );
                    if (argc>3) {
                        if (!(atoi(argv[3])) && strcmp(argv[3],"0")!=0){
                            printf("parameter in this key must be integer value\n");
                            return 0;}
                        colorCount = atoi(argv[3]);
                        if (colorCount>255 || colorCount<0) {printf("parameter must
be >=0 and <=255\n");
                            return 0;}
                        if((strcmp(whichColor, "r") == 0 || strcmp(whichColor, "g")
== 0 || strcmp(whichColor, "b") == 0) && (colorCount == 0 ||
colorCount == 255))
                            componentRBG(whichColor, colorCount);
                        else
                            puts("Color must be r, g or b and value must be 0 or 255");

                    }
                    break;
                case 'i':
                    imageInfo(bmfh, bmih);
                    break;
                case 'h':
                    printMenu();
                    break;
                case '2':
                    if (argc>2) {
                        if (!(atoi(argv[2])) && strcmp(argv[2],"0")!=0){
                            printf("parameter in this key must be integer value\n");
                            return 0;}
                        x = atoi(argv[2]);
                        if (x<0) {printf("parameter must be >=0\n");
                            return 0;}
                    }
                    if (argc>3) {
                        if (!(atoi(argv[3])) && strcmp(argv[3],"0")!=0){
                            printf("parameter in this key must be integer value\n");
                            return 0;}
                        y = atoi(argv[3]);
                        if ( y<0) {printf("parameter must be >=0\n");
                            return 0;}
                    }
                    if (argc>4) {
                        if (!(atoi(argv[4])) && strcmp(argv[4],"0")!=0){
                            printf("parameter in this key must be integer value\n");
                            return 0;}
                        rad = atoi(argv[4]);
                        if ( rad<0) {printf("parameter must be >=0\n");
                            return 0;}
                    }
                    if (argc>5) {
                        if (!(atoi(argv[5]))&& strcmp(argv[5],"0")!=0){

```

```

printf("parameter in this key must be integer value\n");
return 0;}
thick = atoi(argv[5]);
if ( thick<0) {printf("parameter must be >=0\n");
return 0;}
}
if (argc>6) {
if (!(atoi(argv[6])) && strcmp(argv[6],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
b = atoi(argv[6]);
if (b>255 || b<0) {printf("parameter must be >=0 and <=255\
n");
return 0;}
}
if (argc>7) {
if (!(atoi(argv[7])) && strcmp(argv[7],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
g = atoi(argv[7]);
if (g>255 || g<0) {printf("parameter must be >=0 and <=255\
n");
return 0;}
}
if (argc>8) {
if (!(atoi(argv[8])) && strcmp(argv[8],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
r = atoi(argv[8]);
if (r>255 || r<0) {printf("parameter must be >=0 and <=255\
n");
return 0;}
}
if (argc>11) {
if (!(atoi(argv[9])) && strcmp(argv[9],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
fb = atoi(argv[9]);
if (fb>255 || fb<0) {printf("parameter must be >=0 and
<=255\n");
return 0;}
if (!(atoi(argv[10]))&& strcmp(argv[10],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
fg = atoi(argv[10]);
if (fg>255 || fg<0) {printf("parameter must be >=0 and
<=255\n");
return 0;}
if (!(atoi(argv[11]))&& strcmp(argv[11],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
fr = atoi(argv[11]);
if (fr>255 || fr<0) {printf("parameter must be >=0 and
<=255\n");
return 0;}
fill=1;
drawCircle(x,y,rad,thick);

```

```

    }
    else {
        fill=0;
        drawCircle(x,y,rad,thick);
    }
}
break;
case '1':
    if (argc>2) {
        if (!(atoi(argv[2])) && strcmp(argv[2],"0")!=0){
            printf("parameter in this key must be integer value\n");
            return 0;}
        x = atoi(argv[2]);
        if (x<0) {printf("parameter must be >=0\n");
            return 0;}
    }
    if (argc>3) {
        if (!(atoi(argv[3])) && strcmp(argv[3],"0")!=0){
            printf("parameter in this key must be integer value\n");
            return 0;}
        y = atoi(argv[3]);
        if ( y<0) {printf("parameter must be >=0\n");
            return 0;}
    }
    if (argc>4) {
        if (!(atoi(argv[4])) && strcmp(argv[4],"0")!=0){
            printf("parameter in this key must be integer value\n");
            return 0;}
        x1 = atoi(argv[4]);
        if ( x1<0) {printf("parameter must be >=0\n");
            return 0;}
    }
    if (argc>5) {
        if (!(atoi(argv[5]))&& strcmp(argv[5],"0")!=0){
            printf("parameter in this key must be integer value\n");
            return 0;}
        y1 = atoi(argv[5]);
        if ( y1<0) {printf("parameter must be >=0\n");
            return 0;}
    }
    if (argc>6) {
        if (!(atoi(argv[6]))&& strcmp(argv[6],"0")!=0){
            printf("parameter in this key must be integer value\n");
            return 0;}
        thick = atoi(argv[6]);
        if ( thick<0) {printf("parameter must be >=0\n");
            return 0;}
    }
    if (argc>7) {
        if (!(atoi(argv[7])) && strcmp(argv[7],"0")!=0){
            printf("parameter in this key must be integer value\n");
            return 0;}
        b = atoi(argv[7]);
        if (b>255 || b<0) {printf("parameter must be >=0 and <=255\n");
            return 0;}
    }
}
n");

```

```

    }
    if (argc>8) {
    if (!(atoi(argv[8])) && strcmp(argv[8],"0")!=0){
    printf("parameter in this key must be integer value\n");
    return 0;}
    g = atoi(argv[8]);
    if (g>255 || g<0) {printf("parameter must be >=0 and <=255\n");
n");
    return 0;}
    }
    if (argc>9) {
    if (!(atoi(argv[9])) && strcmp(argv[9],"0")!=0){
    printf("parameter in this key must be integer value\n");
    return 0;}
    r = atoi(argv[9]);
    if (r>255 || r<0) {printf("parameter must be >=0 and <=255\n");
n");
    return 0;}
    if (argc>12) {
    if (!(atoi(argv[10])) && strcmp(argv[10],"0")!=0){
    printf("parameter in this key must be integer value\n");
    return 0;}
    fb = atoi(argv[10]);
    if (fb>255 || fb<0) {printf("parameter must be >=0 and
<=255\n");
    return 0;}
    if (!(atoi(argv[11]))&& strcmp(argv[11],"0")!=0){
    printf("parameter in this key must be integer value\n");
    return 0;}
    fg = atoi(argv[11]);
    if (fg>255 || fg<0) {printf("parameter must be >=0 and
<=255\n");
    return 0;}
    if (!(atoi(argv[12]))&& strcmp(argv[12],"0")!=0){
    printf("parameter in this key must be integer value\n");
    return 0;}
    fr = atoi(argv[12]);
    if (fr>255 || fr<0) {printf("parameter must be >=0 and
<=255\n");
    return 0;}
    fill=1;
    drawCircle(x1-(x1-x)/2,y-(x1-x)/2,(x1-x)/2,thick);
    }
    else {
    fill=0;
    drawCircle(x1-(x1-x)/2,y-(x1-x)/2,(x1-x)/2,thick);
    }
    }
break;
case 's':
    if (argc>2) {
    if (!(atoi(argv[2])) && strcmp(argv[2],"0")!=0){
    printf("parameter in this key must be integer value\n");
    return 0;}
    x = atoi(argv[2]);
    if (x<0) {printf("parameter must be >=0\n");

```

```

return 0;}
}
if (argc>3) {
if (!(atoi(argv[3])) && strcmp(argv[3],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
y = atoi(argv[3]);
if ( y<0) {printf("parameter must be >=0\n");
return 0;}
}
if (argc>4) {
if (!(atoi(argv[4])) && strcmp(argv[4],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
rad = atoi(argv[4]);
if ( rad<0) {printf("parameter must be >=0\n");
return 0;}
}
if (argc>5) {
if (!(atoi(argv[5]))&& strcmp(argv[5],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
thick = atoi(argv[5]);
if ( thick<0) {printf("parameter must be >=0\n");
return 0;}
}
if (argc>6) {
if (!(atoi(argv[6])) && strcmp(argv[6],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
b = atoi(argv[6]);
if (b>255 || b<0) {printf("parameter must be >=0 and <=255\
n");
return 0;}
}
if (argc>7) {
if (!(atoi(argv[7])) && strcmp(argv[7],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
g = atoi(argv[7]);
if (g>255 || g<0) {printf("parameter must be >=0 and <=255\
n");
return 0;}
}
if (argc>8) {
if (!(atoi(argv[8])) && strcmp(argv[8],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}
r = atoi(argv[8]);
if (r>255 || r<0) {printf("parameter must be >=0 and <=255\
n");
return 0;}
if (argc>11) {
if (!(atoi(argv[9])) && strcmp(argv[9],"0")!=0){
printf("parameter in this key must be integer value\n");
return 0;}

```

```

        fb = atoi(argv[9]);
        if (fb>255 || fb<0) {printf("parameter must be >=0 and
<=255\n");
        return 0;}
        if (!(atoi(argv[10]))&& strcmp(argv[10],"0")!=0){
        printf("parameter in this key must be integer value\n");
        return 0;}
        fg = atoi(argv[10]);
        if (fg>255 || fg<0) {printf("parameter must be >=0 and
<=255\n");
        return 0;}
        if (!(atoi(argv[11]))&& strcmp(argv[11],"0")!=0){
        printf("parameter in this key must be integer value\n");
        return 0;}
        fr = atoi(argv[11]);
        if (fr>255 || fr<0) {printf("parameter must be >=0 and
<=255\n");
        return 0;}
        fill=1;
        drawSquare(x,y,rad,thick);
    }
    else {
        fill=0;
        drawSquare(x,y,rad,thick);
    }
}
break;
case 't':

    if (argc>2) {
        if (!(atoi(argv[2])) && strcmp(argv[2],"0")!=0){
        printf("parameter in this key must be integer value\n");
        return 0;}
        x = atoi(argv[2]);
    }
    if (argc>3) {
        if (!(atoi(argv[3])) && strcmp(argv[3],"0")!=0){
        printf("parameter in this key must be integer value\n");
        return 0;}
        y = atoi(argv[3]);
    }
    if (argc>4) {
        if (!(atoi(argv[4])) && strcmp(argv[4],"0")!=0){
        printf("parameter in this key must be integer value\n");
        return 0;}
        x1 = atoi(argv[4]);
    }
    if (argc>5) {
        if (!(atoi(argv[5]))&& strcmp(argv[5],"0")!=0){
        printf("parameter in this key must be integer value\n");
        return 0;}
        y1 = atoi(argv[5]);
    }
    if (argc>6) {
        if (!(atoi(argv[6]))&& strcmp(argv[6],"0")!=0){
        printf("parameter in this key must be integer value\n");

```



```

        return 0;}
        corner = atoi(argv[6]);
        turn(x,y,x1,y1,corner);
    }

    break;
}
opt = getopt_long( argc, argv, optString, longOpts, &longIndex );
}
if (flag == 0) for(int i = 0; i < bmih.biHeight; i++)
    free(arr[i]);
    if (flag == 1) for(int i = 0; i < bmih.biWidth; i++)
        free(arr[i]);
    free(arr);

    free(pad);
    free(inputName);
    free(outputName);
}

```