МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP-изображений

Студент гр. 9303 _____ Максимов Е.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Максимов Е.А.

Группа 9303

Тема работы: Обработка BMP изображений

Исходные данные:
BMP-изображение, набор ключей и аргументов

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

Предполагаемый объем пояснительной записки:
32 страницы

Дата выдачи задания:

Дата сдачи работы: 17.06.2020

Дата защиты работы:

Студент _____ Максимов Е.А

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

В данной работе представлена программа на языке программирования C, которая производит обработку BMP-изображения посредством набора методов класса. Программе на вход подаются ключи и аргументы к ключам, которые определяют исходное изображение и действиями с ним. Программа обрабатывает все входящие аргументы, загружает в динамическую память данные изображения и далее оперирует только с ним.

Далее, программа делает одно или несколько следующих действий, согласно полученным ключам и аргументам.

1) Рисует окружность определённого радиуса (или вписанную в квадрат, заданный координатами), толщины и цвета; если присутствует необходимый ключ и аргумент к нему, то внутри окружность может также быть закрашена.
2) Отражает заданную координатами область по горизонтали или по вертикали.
3) Копирует заданную координатами область в другую область изображения.

Программа корректно работает с файлами, которые удовлетворяют следующим условиям:

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является изучение структуры BMP-изображения и разработка программы на языке C++, которая считывает и редактирует изображение по заранее заданным ключам и аргументам.

Для достижения поставленной цели требуется решить следующие задачи:

1) изучение структуры файла формата BMP для реализации функций для работы с данном типом данных и функциями библиотеки *getopt.h* для обработки команд пользователя;

2) разработка программного кода;

3) сборка программы из полученного программного кода;

4) тестирование программного кода.

1. ОПИСАНИЕ КОДА ПРОГРАММЫ. СТРУКТУРЫ

1.1. Заголовок верхнего уровня BMP-изображения BMPFileHeader.

```
typedef struct{
    unsigned short signature;
    unsigned int sizeFile;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int offset;
}BMPFileHeader;
```

Поля структуры представляют собой набор основных данных BMP-изображения:

• сигнатура файла, *unsigned short signature*;

• размер изображения, *unsigned int sizeFile*;

• два зарезервированных поля, *unsigned short reserved1, unsigned short reserved2*;

• смещение от поля данных в байтах, *unsigned int offset*.

Поля данной структуры упакованы в памяти директивой *pragma pack*.

1.2. Заголовок нижнего уровня BMP-изображения BMPInfoHeader.

```
typedef struct{
    unsigned int sizeHeader;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bitPerPixel;
    unsigned int compression;
    unsigned int sizeImage;
    unsigned int xPerMeter;
    unsigned int yPerMeter;
    unsigned int colorTable;
    unsigned int colorCount;
}BMPInfoHeader;
```

Поля структуры представляют собой набор дополнительных данных BMP-изображения, определяющих свойства изображения:

• размер данной структуры в байтах, *unsigned int sizeHeader*;

• ширина изображения, *unsigned int width*;

• высота изображения, *unsigned int height*;

• количество слоёв (плоскостей) в изображении, *unsigned short planes*;

• количество бит на точку, *unsigned short bitPerPixel*;

• тип сжатия изображения, *unsigned int compression*;

• размер изображения в байтах, *unsigned int sizeImage*;

• количество пикселей на метр по ширине, *unsigned int xPerMeter*;

• количество пикселей на метр по высоте, *unsigned int yPerMeter*;

• количество цветов в таблице цветов, если она есть, *unsigned int colorTable*;

• количество «важных» цветов, *unsigned int colorCount*.

Поля данной структуры упакованы в памяти директивой *pragma pack*.

1.3. Цвет точки изображения Pixel.

```
typedef struct{
    unsigned char B;
    unsigned char G;
    unsigned char R;
}Pixel;
```

Поля структуры представляют собой набор из 3 чисел, кодирующих цвет точки в формате RGB:

• уровень канала синего цвета, *unsigned char B;*

• уровень канала зелёного цвета, *unsigned char G*;

• уровень канала красного цвета, *unsigned char R*.

Расположение полей в обратном порядке связано с размещением данных в файлах формата BMP.

Поля данной структуры упакованы в памяти директивой *pragma pack*.

1.4. Координаты точки изображения Point.

```
typedef struct{
    int x;
    int y;
}Point;
```

Поля структуры представляют собой геометрические координаты точки на полотне изображения.

• координата по ширине, *int x*;

• координата по высоте, *int y*.

Координаты отчитываются от левого верхнего угла изображения; ширина по горизонтали, высота по вертикали.

2. ОПИСАНИЕ КОДА ПРОГРАММЫ. КЛАССЫ И МЕТОДЫ

2.1. Класс пользовательских параметров Parameters.

Класс состоит из большого количества свойств, большинство из которых имеет стандартные значения. Свойства принимают значения согласно полученным ключам и аргументам. В дальнейшем все полученные данные передаются в качестве аргументов к методам класса *BMPImage* (см. ниже).

2.1.1. Свойства класса Parameters.

Класс не имеет приватных или защищённых свойств.

Класс имеет следующие публичные свойства:

1) общие свойства:

• *char loadFile[256]* – название загружаемого файла; по умолчанию {'\0'};

• *bool loadFileF* – флаг указания загружаемого файла; по умолчанию 0;

• *char saveFile[256]* – название сохраняемого файла; по умолчанию "out.bmp";

• *bool infoF* – флаг указания сохраняемого файла; по умолчанию 0

• *int queue[8]* – набор переменных, показывающих очерёдность выполнения операций над изображением; по умолчанию {0};

2) свойства для рисования окружности:

• *bool circleF* – флаг использования функции рисования окружности; по умолчанию 0;

• *int circleType* – тип рисования окружности (1 – при помощи центра окружности, 2 – при помощи описанного квадрата); по умолчанию 0;

• *Point circleA* – точка левого верхнего угла квадрата;

• *Point circleB* – точка правого нижнего угла квадрата;

• *Point circleO* – точка центра окружности;

• *int circleRadius* – радиус окружности; по умолчанию 0;

• *int circleThickness* – толщина окружности; по умолчанию 0;

• *bool circleColorCircleF* – флаг цвета окружности; по умолчанию 0;

• *Pixel circleColorCircle* – цвет окружности; по умолчанию {0, 0, 0};

• *bool circleColorFill* – флаг цвета заливки; по умолчанию 0;

• *Pixel circleColorCircle* – цвет заливки; по умолчанию {0, 0, 0};

3) свойства для отражения заданной области:

• *bool mirrorF* – флаг использования функции отражения области; по умолчанию 0;

• *Point mirrorA* – координаты верхней левой точки области отражения*;*

• *Point mirrorB* – координаты нижней правой точки области отражения*;*

4) свойства для копирования заданной области:

• *int mirrorType* – тип отражения (по горизонтали или вертикали); по умолчанию 0;

• *bool copyF* – флаг использования функции копирования области; по умолчанию 0;

• *Point copyA* – координаты верхней левой точки источника копирования*;*

• *Point copyB* – координаты нижней правой точки источника копирования*;*

• *Point copyC* – координаты верхней левой точки вставки буфера копирования.

2.1.2. Методы класса Parameters.

Класс имеет следующие методы:

1) *void queueAdd(int n).*

```
void queueAdd(int n){
      for(int i=0; i<8; i++) if(queue[i]==0){
          queue[i] = n;
          return;
      }
}
```

Принимает на вход переменную целого типа *n* – требуемая операция над изображением, и заменяет первое нулевое число в массиве *queue* на *n*. Метод не имеет возвращаемого значения.

2.2 Класс BMP-изображения BMPImage.

Основной класс программы, в котором хранится полученное BMP-изображение. По большей части состоит из методов, с помощью которых происходит работа с BMP-изображением.

2.2.1. Свойства класса BMPImage.

Класс имеет следующие приватные свойства:

• *BMPFileHeader BFH* – заголовок верхнего уровня BMP-изображения;

• *BMPInfoHeader BIH* – заголовок нижнего уровня BMP-изображения;

• *Pixel** PixelArray* – указатель на двумерный массив точек изображения; по умолчанию NULL;

• *char* filename[256]* – указатель на строку, содержащую путь к изображению.

2.2.2. Методы класса BMPImage.

Класс имеет следующие методы:

1) Деструктор.

```
~BMPImage(){
    if(PixelArray!=NULL){
        for(int i=0; i<BIH.height; i++) if (PixelArray[i] != NULL) free(P
ixelArray[i]);
        free(PixelArray);
    }
}
```

Освобождает выделенную для хранения двумерного массива точек изображения *PixelArray* в том случае, если он был использован для хранения данных.

2) checkHeader().

```
int checkHeader(){
    if ((BFH.signature != 0x4d42) && (BFH.signature != 0x4349) && (BFH.si
gnature != 0x5450)) return 1;
    if (BIH.bitPerPixel != 24) return 1;
    if (BIH.compression != 0) return 1;
    if ((BIH.width > 8192) || (BIH.height > 8192)) return 2;
    return 0;
}
```

Проверяет данные, записанные в поля структур заголовков верхнего и нижнего уровня (в точности сигнатуру, количество бит на точку, уровень сжатия изображения, ширину и высоту изображения.

Возвращает 0, если данные прошли проверку; 2, если размер изображения превышает установленный лимит; 1 в остальных случаях провала проверки.

3) int checkPoint(Point A).

```
int checkPoint(Point A){
    if( (A.x < 0) || (A.y < 0) || (A.x >= BIH.width) || (A.y >= BIH.heigh
t) ) return 1;
    return 0;
}
```

Принимает на вход координаты *Point A* и проверяет, принадлежат ли они заданному полотну изображения по ширине и высоте. Возвращает 0, если проверка прошла успешно, и 1 в случае провала проверки.

4) *void load(const char* name).* (см. приложение А)

Принимает на вход строку *name* – путь к изображению. Метод проверяет, возможно ли открыть изображение, которое расположено по данному пути. В случае успеха считывает его в двумерный динамический массив точек изображения *PixelArray*, в противном случае печатает на экран сообщение об ошибке и останавливает дальнейший ход программы.

Метод записывает в массив точки изображения построчно, но в обратном порядке. Это связано с размещением данных в файлах формата BMP.

Переменная *padding* необходима для нахождения количества нулевых байтов, которые расположены в конце каждой строки точек. Переменная рассчитывается так, чтобы сумма байтов в каждой строке была кратна 4. Это связано с размещением данных в файлах формата BMP.

5) *void info().*

```
void info(){
    printf("\n[BMP] Name:\t\t%s\n", filename);
    printf("[BMP] Signature:\t%x\n", BFH.signature);
```

```
    printf("[BMP] Size:\t\t%u bytes\n", BFH.sizeFile);
    printf("[BMP] Resolution:\t%u x %u\n", BIH.width, BIH.height);
    printf("[BMP] Bits per pixel:\t%u\n\n", BIH.bitPerPixel);
    return;
}
```

Печатает на экран данные о полученном изображении.

6) Pixel getColor(char color)* (см. приложение А)

Принимает на вход строку *color* – название одного из следующих цветов: *red, orange, yellow, lime, green, turquoise, aqua, blue, purple, pink, brown, white, grey, black.* Возвращает соответствующий цвет в качестве переменной цветной точки *Pixel* (RGB). В случае несовпадения, возвращает точку, соответствующую чёрному цвету (black).

7) void drawCircle(Point O, int R, int thickness, Pixel colorCircle, int fillF, Pix el colorFill) (см. приложение А)

Принимает на вход следующие данные: координаты центра окружности *Point O*, радиус *int R*, толщина *int t*, цвет окружности *Pixel ColorCircle*, а также флаг заливки окружности *inf fillF* и цвет заливки *Pixel colorFill*. Метод заменяет цвета определённых точек из двумерного массива *PixelArray*. Метод действует согласно алгоритму Брезенхема. Для рисования окружностей с толщиной, больше чем 1, алгоритм рисует окружности меньшего радиуса.

В случае, если точка изображения не принадлежит окружности, метод пропускает её. Это допускает возможность рисовать окружности, расположенные на изображении частично.

8) void copy(Point O1, Point O2, Point O3).

```
void copy(Point O1, Point O2, Point O3){
    Pixel tempPixelArray[O2.y-O1.y+1][O2.x-O1.x+1];
    for(int i=0; i<=O2.y-O1.y; i++) for(int j=0; j<=O2.x-
O1.x; j++) tempPixelArray[i][j] = PixelArray[i+O1.y][j+O1.x];
    for(int i=0; i<=O2.y-O1.y; i++) for(int j=0; j<=O2.x-O1.x; j++){
        if(O3.y+i>=BIH.height) break;
        if(O3.x+j>=BIH.width) continue;
        PixelArray[O3.y+i][O3.x+j] = tempPixelArray[i][j];
    }
```

```
    if((O2.y-O1.y+O3.y>=BIH.height)||(O2.x-
O1.x+O3.x>=BIH.width)) printf(WARNING_NO_SPACE);
    return;
}
```

Принимает на вход следующие данные: координаты левого верхнего и правого нижнего участков копирования *Point O1* и *Point O2* соответственно и координаты точки вставки *Point O3*. Метод записывает в буферный двумерный массив данные для копирования и вставляет их, начиная с указанной точки вставки.

Если в данной строке точек недостаточно места, то копирование прерывается на данной строке, и происходит операция со следующей строкой; если строки точек, с которой производится операция, не существует, то метод прерывает свою работу. В том или ином случае метод печатает на экран предупреждение (warning) об этом. Это позволяет копировать участки изображения и вставлять их на краях изображения.

9) void mirror(Point O1, Point O2, int type).

```
void mirror(Point O1, Point O2, int type){
    Pixel temp;
    if(type == 1){
        for(int i=O1.y; i<=O2.y; i++) for(int j=O1.x; j<(O1.x+O2.x)/2; j+
+){
            temp = PixelArray[i][j];
            PixelArray[i][j] = PixelArray[i][O1.x+O2.x-j];
            PixelArray[i][O1.x+O2.x-j] = temp;
        }
    } else if(type == 2){
        for(int j=O1.x; j<=O2.x; j++) for(int i=O1.y; i<(O1.y+O2.y)/2; i+
+){
            temp = PixelArray[i][j];
            PixelArray[i][j] = PixelArray[O1.y+O2.y-i][j];
            PixelArray[O1.y+O2.y-i][j] = temp;
        }
    }
    return;
}
```

Принимает на вход координаты левого верхнего и правого нижнего участков отражения *Point O1* и *Point O2* соответственно и тип отражения *int type*. С помощью буферной переменной *Pixel temp* метод производит последовательный обмен точек изображения. В зависимости от типа отражения

(который указывается с помощью ключа *axis*), участок изображения будет отражён горизонтально или вертикально.

10) *void save(const char* name).* (см. приложение А)

Принимает на вход строку *name* – имя сохраняемого файла. В случае успешного доступа, метод создаёт файл и записывает в него данные о BMP-изображении: заголовки изображения и массив точек.

Метод записывает в файл точки изображения построчно, но в обратном порядке. Это связано с размещением данных в файлах формата BMP.

11) *void help().*

```
void help(){
   printf("[BMP] --load|-l <filename>, --save|-s <filename> -
 load or save BMP image.\n");
    printf("[BMP] --info|-i - show information about BMP image.\n");
    printf("[BMP] --circle <x1,y1>|<x2,y2,x3,y3> -
 draw circle centered at x1,y1 or inscribed in square at x2,y2,x3,y3. Cen
tered circle don't requires radius.\n");
    printf("[BMP] --radius|-r <R> - set radius R of centered circle; --
thickness|-t <T> - set thickness T of circle.\n");
    printf("[BMP] --color|-c <color> - set color of circle; --fill|-
f <color> - set color of filling, optional.\n");
    printf("[BMP] Available colors: red, orange, yellow, lime, green, tur
quoise, aqua, blue, purple, pink, brown, white, gray, black.\n");
    printf("[BMP] --mirror <x1,y1,x2,y2> -
 mirror part of image at x1,y1,x2,y2.\n");
    printf("[BMP] --axis|-a h|v -
 type of mirror (horizontally or vertically).\n");
    printf("[BMP] --copy <x1,y1,x2,y2,x3,y3> -
 copy part of image from x1,y1,x2,y2 to x3,y3.\n");
    printf("[BMP] --help|h -
 show this message. Use this key standalone.\n\n");
    return;
}
```

Печатает на экран сообщение-справку.

3. ОПИСАНИЕ КОДА ПРОГРАММЫ. ОСНОВНАЯ ФУНКЦИЯ.

Основная функция *int main(int argc, char* argv[])* принимает от пользователя набор ключей и аргументов к ним. При помощи подключённой библиотеки *getopt.h* функция осуществляет перебор получаемых данных.

```
  struct option longOpts[]={
        {"load", required_argument, NULL, 'l'},
        {"save", required_argument, NULL, 's'},
        {"help", no_argument, NULL, 'h'},
        {"info", no_argument, NULL, 'i'},
        {"radius", required_argument, NULL, 'r'},
        {"thickness", required_argument, NULL, 't'},
        {"color", required_argument, NULL, 'c'},
        {"fill", required_argument, NULL, 'f'},
        {"axis", required_argument, NULL, 'a'},
        {"circle", required_argument, NULL, 1},
        {"mirror", required_argument, NULL, 2},
        {"copy", required_argument, NULL, 3},
        {NULL, 0, NULL, 0},
};
```

Структура *option* (определена в библиотеке *getopt.h*) хранит названия длинных ключей, флаг аргумента к ключу, тип возвращаемого значения и возвращаемое значение из функции *getopt_long*.

```
const char opts[] = "l:s:hir:t:c:f:a:";
```

В данной строке указаны посимвольно доступные короткие ключи; двоеточие после символа означает, что данный ключ требует аргумент.

В структуре выше определены следующие длинные ключи, для которых определены следующие аргументы:

1) *circle* – рисование окружности; принимает 1 аргумент:

• строка – координаты через запятую без пробелов: для окружности с центром – x1,y1, для вписанной в квадрат окружности – x1,y1,x2,y2. Ожидаются неотрицательные целые числа в качестве аргументов.

Требует среди входных данных аргументы для следующих ключей: *thickness, color.* Если была введена 1 пара координат, то требует аргумент для ключа *radius*.

Опциональные аргументы: *fill.*

2) *mirror* – отражение заданного участка изображения; принимает 1 аргумент:

• строка – координаты через запятую без пробелов: x1,y1,x2,y2.

Ожидаются неотрицательные целые числа в качестве аргументов.

Требует среди входных данных аргументы для следующих ключей: *axis*.

3) *copy* – копирование заданного участка изображения принимает 1 аргумент:

• строка – координаты через запятую без пробелов: x1,y1,x2,y2,x3,y3.

Ожидаются неотрицательные целые числа в качестве аргументов.

Также определены ключи, которые могут быть введены пользователем в длинной или короткой форме:

1) *load, l* – загрузка файла; принимает 1 аргумент:

• строка – путь к исходному файлу;

2) *save, s* – сохранение файла; принимает 1 аргумент:

• строка – название нового файла;

3) *help, h* – вызов справки;

4) *info, i* – печать информации о загруженном файле;

5) *radius, r* – радиус окружности; принимает 1 аргумент:

• целое неотрицательное число – радиус окружности;

6) *thickness, t* – толщина окружности; принимает 1 аргумент:

• целое неотрицательное число – толщина окружности;

7) *color, c* – цвет окружности; принимает 1 аргумент:

• строка – название цвета;

8) *fill, f* – цвет заливки окружности; принимает 1 аргумент:

• строка – название цвета.

9) *axis, a* – тип отражения: принимает 1 аргумент:

• символ – *h* для отражения по горизонтали, *v* – по вертикали.

Конструкция *case-switch* с помощью цикла *while* перебирает ключи и аргументы к ним. В цикле присутствует большое множество проверок для вводимых пользователем данных; проверки необходимы для корректной работы программы (проверка на количество поступающих аргументов, корректность введённых аргументов и прочее). В случае обнаружения любого некорректного использования ключа, программа печатает сообщение об ошибке и завершает работу.

Программа проверяет поступающее изображение с помощью метода *checkHeader*. Программа работает с изображениями, соответствующие следующим условиям:

• формат изображения: BMP;

• сигнатура изображения: *4d32, 4349* или *5450*;

• количество битов на точку изображения: 24;

• тип сжатия изображения: 0;

• размер полотна: не должен превышать 8192 по ширине или высоте.

При поступлении ключей *circle, mirror* или *copy*, в массив *queue* добавляется число 1, 2 или 3 – соответствующий номер операции. При успешной проверке всех поступивших аргументов, программа с помощью другого блока *switch-case* выполняет операции в порядке, указанным пользователем.

В случае, если пользователь ввёл один из ключей *circle, mirror* или *copy,* но не указал название сохраняемого файла, программа сохранит его с названием *out.bmp*.

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

```
root@ZhenjaMax:~/C# ./a.out -l marbles.bmp -h --info
[BMP] --load|-l <filename>, --save|-s <filename> - load or save BMP image.
[BMP] --info|-i - show information about BMP image.
[BMP] --circle <x1,y1>|<x2,y2,x3,y3> - draw circle centered at x1,y1 or inscribed in square at x2,y2,x3,y3. Centered circle don't requires radius.
[BMP] --radius|-r <R> - set radius R of centered circle; --thickness|-t <T> - set thickness T of circle.
[BMP] --color|-c <color> - set color of circle; --fill|-f <color> - set color of filling, optional.
[BMP] Available colors: red, orange, yellow, lime, green, turquoise, aqua, blue, purple, pink, brown, white, gray, black.
[BMP] --mirror <x1,y1,x2,y2> - mirror part of image at x1,y1,x2,y2.
[BMP] --axis|-a h|v - type of mirror (horizontally or vertically).
[BMP] --copy <x1,y1,x2,y2,x3,y3> - copy part of image from x1,y1,x2,y2 to x3,y3.
[BMP] --help|h - show this message. Use this key standalone.

[BMP] File marbles.bmp opened successfully.

[BMP] Name:             marbles.bmp
[BMP] Signature:        4d42
[BMP] Size:             4264314 bytes
[BMP] Resolution:       1419 x 1001
[BMP] Bits per pixel:   24

root@ZhenjaMax:~/C#
```

Рис. 1. Демонстрация работы методов *help, info*.

```
root@ZhenjaMax:~/C# ./a.out -l file1.qwerty -s file2.qwerty
[BMP] Error! Unable to open file file1.qwerty.
root@ZhenjaMax:~/C#
```

Рис. 2. Демонстрация работы программы для некорректного ввода.

Рис. 3. Демонстрация работы программы для ввода *«./a.out -l white.bmp --circle 40,40,120,120 -t 30 -c orange -f black -s out1.bmp»* (подзадача №1).

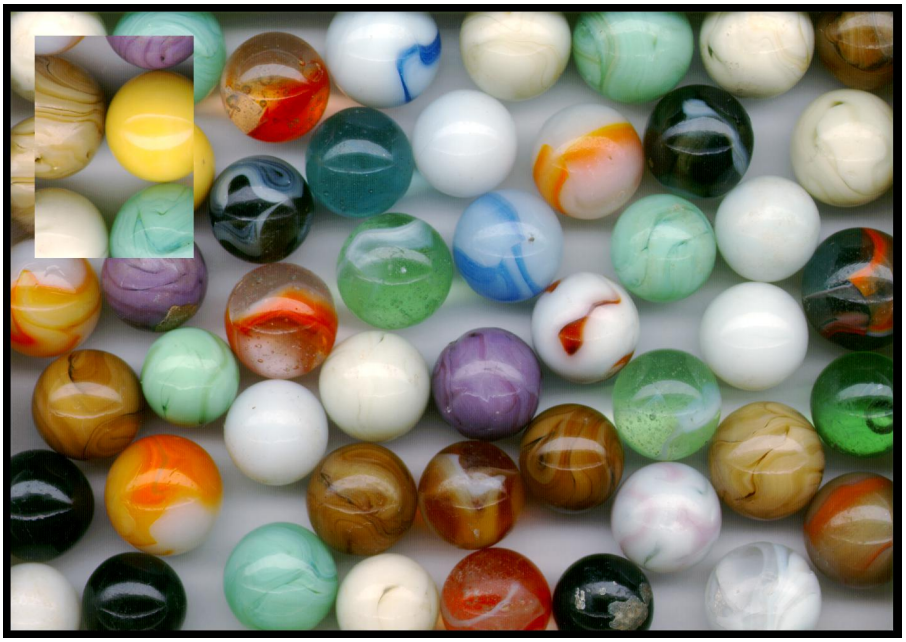

Рис. 4. Демонстрация работы программы для ввода *«./a.out --load marbles.bmp -s out2.bmp --mirror 50,50,300,400 --axis v»* (подзадача №2).

Рис. 5. Демонстрация работы программы для ввода *«./a.out --save out3.bmp -l night.bmp --copy 200,250,300,400,400,400»* (подзадача №3).

Рис. 6. Демонстрация работы программы для ввода *«./a.out --save out4.bmp --circle 200,200 --radius 40 -c lime --thickness 10 -l night.bmp --mirror 200,160,240,240 -a v --copy 160,160,240,240,240,240»* (комбинация).

ЗАКЛЮЧЕНИЕ

В ходе работы были изучена структура файлов формата BMP и библиотека *geopt.h* для работы с ключами и аргументами пользователя. В результате работы была разработана программа, которая считывает BMP-изображение и редактирует его по заранее заданным ключам и аргументам. Для каждой подзадачи были описаны классы и их методы для корректной работы программы. Сопоставленные входные и выходные данные программы удовлетворяют условию задания.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Справочник «Man-7» // https://man7.org/linux/man-pages/man3/getopt.3.html#top_of_page (дата обращения: 11.05.2020 г.)

2. Онлайн-энциклопедия // Wikipedia // https://ru.wikipedia.org/wiki/BMP (дата обращения: 11.05.2020 г.)

3. Онлайн-справочник языков программирования // W3Schools // https://www.w3schools.com/cpp/cpp_classes.asp (дата обращения: 11.05.2020 г.)

4. Онлайн-энциклопедия // Wikipedia // https://ru.wikipedia.org/wiki/Алгоритм_Брезенхэма (дата обращения: 11.05.2020 г.)

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <iostream>
#include <cstring>
#include <getopt.h>

#define ERR_NO_FILE "[BMP] Error! Please, choose an image.\n"
#define ERR_FILE "[BMP] Error! An error occurred while processing the i
mage.\n"
#define ERR_FEW_ARGS "[BMP] Error! Not enough arguments. Please, use -
h for help.\n"
#define ERR_SAME_ARGS "[BMP] Error! There are more than one argument of
 same type.\n"
#define ERR_WRONG_ARGS "[BMP] Error! Incorrect arguments. Use -
h for help.\n"
#define WARNING_COLOR "[BMP] Warning! Incorrect argument of color. Set
the default color to black.\n"
#define WARNING_SAVE "[BMP] Warning! No arguments for save. Set the def
ault out.bmp.\n"
#define WARNING_NO_SPACE "[BMP] Warning! Not enough space for paste. So
me data will be lost.\n"

using namespace std;

#pragma pack (push, 1)
typedef struct{
    unsigned short signature;
    unsigned int sizeFile;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int offset;
}BMPFileHeader;
typedef struct{
    unsigned int sizeHeader;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bitPerPixel;
    unsigned int compression;
    unsigned int sizeImage;
    unsigned int xPerMeter;
    unsigned int yPerMeter;
    unsigned int colorTable;
    unsigned int colorCount;
}BMPInfoHeader;
typedef struct{
    unsigned char B;
    unsigned char G;
    unsigned char R;
}Pixel;
#pragma pack (pop)
typedef struct{
    int x;
    int y;
```

```
}Point;

class Parameters{
public:
    char loadFile[256] = {'\0'};
    bool loadFileF = 0;
    char saveFile[256] = "out.bmp";
    bool saveFileF = 0;
    bool infoF = 0;
    int queue[8] = {0};

    bool circleF = 0;
    int circleType = 0;
    Point circleA;
    Point circleB;
    Point circleO;
    int circleRadius = 0;
    int circleThickness = 0;
    bool circleColorCircleF = 0;
    Pixel circleColorCircle = {0, 0, 0};
    bool circleColorFillF = 0;
    Pixel circleColorFill = {0, 0, 0};

    bool copyF = 0;
    Point copyA;
    Point copyB;
    Point copyC;

    bool mirrorF = 0;
    Point mirrorA;
    Point mirrorB;
    int mirrorType = 0;

    void queueAdd(int n){
        for(int i=0; i<8; i++) if(queue[i]==0){
            queue[i] = n;
            return;
        }
    }
};

class BMPImage{
private:
    BMPFileHeader BFH;
    BMPInfoHeader BIH;
    Pixel** PixelArray = NULL;
    char filename[256];
public:
    ~BMPImage(){
        if(PixelArray!=NULL){
            for(int i=0; i<(int)BIH.height; i++) if (PixelArray[i] != N
ULL) free(PixelArray[i]);
            free(PixelArray);
        }
    }
    int checkHeader(){
```

```
        if ((BFH.signature != 0x4d42) && (BFH.signature != 0x4349) && (
BFH.signature != 0x5450)) return 1;
        if (BIH.bitPerPixel != 24) return 1;
        if (BIH.compression != 0) return 1;
        if ((BIH.width > 8192) || (BIH.height > 8192)) return 2;
        return 0;
    }
    int checkPoint(Point A){
        if( (A.x < 0) || (A.y < 0) || (A.x >= (int)BIH.width) || (A.y >
= (int)BIH.height) ) return 1;
        return 0;
    }
    void load(const char* name){
        strcpy(filename, name);
        FILE *f = fopen(filename, "rb");
        if(f == NULL){
            printf("[BMP] Error! Unable to open file %s.\n", filename);
            exit(1);
        }
        fread(&BFH, sizeof(BMPFileHeader), 1, f);
        fread(&BIH, sizeof(BMPInfoHeader), 1, f);
        if(checkHeader()==1){
            printf("[BMP] Error! Wrong version of file.\n");
            exit(1);
        } else if(checkHeader()==2){
            printf("[BMP] Error! Resolution of current file is too larg
e! (%u x %x)\n", BIH.width, BIH.height);
            exit(1);
        }
        if(fseek(f, BFH.offset, SEEK_SET)!=0){
            printf(ERR_FILE);
            exit(1);
        }
        int padding = BIH.width%4;
        unsigned char* temp;
        if(padding) temp = (unsigned char*)calloc(padding, sizeof(unsig
ned char));
        if((PixelArray = (Pixel**)malloc(BIH.height*sizeof(Pixel*))) ==
 NULL){
            printf(ERR_FILE);
            exit(1);
        }
        for(int i=0; i<(int)BIH.height; i++) if((PixelArray[i] = (Pixel
*)malloc(BIH.width*sizeof(Pixel))) == 0){
            printf(ERR_FILE);
            exit(1);
        };
        for(int i=BIH.height-1; i>=0; i--){
            if(fread(PixelArray[i], sizeof(Pixel), BIH.width, f) != BIH.
width){
                printf(ERR_FILE);
                exit(1);
            }
            if(padding) fread(temp, sizeof(unsigned char), padding, f);
        }
        free(temp);
        fclose(f);
```

```
        printf("[BMP] File %s opened successfully.\n", filename);
        return;
    }
    void info(){
        printf("\n[BMP] Name:\t\t%s\n", filename);
        printf("[BMP] Signature:\t%x\n", BFH.signature);
        printf("[BMP] Size:\t\t%u bytes\n", BFH.sizeFile);
        printf("[BMP] Resolution:\t%u x %u\n", BIH.width, BIH.height);
        printf("[BMP] Bits per pixel:\t%u\n\n", BIH.bitPerPixel);
        return;
    }
    Pixel getColor(char* color){
        if(!strcmp(color, "red")) return {0, 0, 255};
        if(!strcmp(color, "orange")) return {0, 127, 255};
        if(!strcmp(color, "yellow")) return {0, 255, 255};
        if(!strcmp(color, "lime")) return {0, 255, 127};
        if(!strcmp(color, "green")) return {0, 255, 0};
        if(!strcmp(color, "turquoise")) return {127, 255, 0};
        if(!strcmp(color, "aqua")) return {255, 255, 0};
        if(!strcmp(color, "blue")) return {255, 0, 0};
        if(!strcmp(color, "purple")) return {255, 0, 127};
        if(!strcmp(color, "pink")) return {127, 63, 255};
        if(!strcmp(color, "brown")) return {0, 63, 127};
        if(!strcmp(color, "white")) return {255, 255, 255};
        if(!strcmp(color, "grey")) return {127, 127, 127};
        if(!strcmp(color, "black")) return {0, 0, 0};
        printf(WARNING_COLOR);
        return {0, 0, 0};
    }
    void drawCircle(Point O, int R, int thickness, Pixel colorCircle, i
nt fillF, Pixel colorFill){
        int dx, dy, d, error;
        bool xmax, xmin, ymax, ymin, ymax1, ymin1;
        int thicknessIter = thickness;
        for(; thicknessIter>0; thicknessIter--){
            dx = 0;
            dy = R;
            d = 1 - 2*R;
            error = 0;
            while(dy>=0){
                xmax = O.x+dx<(int)BIH.width;
                xmin = O.x-dx>=0;
                ymax = O.y+dy<(int)BIH.height;
                ymin = O.y-dy>=0;
                ymax1 = O.y+dy+1<(int)BIH.height;
                ymin1 = O.y-dy-1>=0;
                if((ymax)&&(xmax)) PixelArray[O.y+dy][O.x+dx] = colorCi
rcle;
                if((ymax)&&(xmin)) PixelArray[O.y+dy][O.x-
dx] = colorCircle;
                if((ymin)&&(xmax)) PixelArray[O.y-
dy][O.x+dx] = colorCircle;
                if((ymin)&&(xmin)) PixelArray[O.y-dy][O.x-
dx] = colorCircle;
                if(thicknessIter != thickness){
                    if((ymax1)&&(xmax)) PixelArray[O.y+dy+1][O.x+dx] =
colorCircle;
```

```
                    if((ymax1)&&(xmin)) PixelArray[O.y+dy+1][O.x-
dx] = colorCircle;
                    if((ymin1)&&(xmax)) PixelArray[O.y-dy-
1][O.x+dx] = colorCircle;
                    if((ymin1)&&(xmin)) PixelArray[O.y-dy-1][O.x-
dx] = colorCircle;
                }
                if((fillF)&&(thicknessIter==1)) for(int i=O.y-
dy+1; i<O.y+dy; i++){
                    if(i<0) continue;
                    if(i>=(int)BIH.height) break;
                    if(xmin) PixelArray[i][O.x-dx] = colorFill;
                    if(xmax) PixelArray[i][O.x+dx] = colorFill;
                }
                error = 2*(d+dy)-1;
                if((d<0)&&(error<0)){
                    d+=(2*++dx + 1);
                    continue;
                }
                if((d>0)&&(error>0)){
                    d-=(2*--dy + 1);
                    continue;
                }
                d += 2*(++dx - --dy);
            }
            R--;
        }
        return;
    }
    void copy(Point O1, Point O2, Point O3){
        Pixel tempPixelArray[O2.y-O1.y+1][O2.x-O1.x+1];
        for(int i=0; i<=O2.y-O1.y; i++) for(int j=0; j<=O2.x-
O1.x; j++) tempPixelArray[i][j] = PixelArray[i+O1.y][j+O1.x];
        for(int i=0; i<=O2.y-O1.y; i++) for(int j=0; j<=O2.x-
O1.x; j++){
            if(O3.y+i>=(int)BIH.height) break;
            if(O3.x+j>=(int)BIH.width) continue;
            PixelArray[O3.y+i][O3.x+j] = tempPixelArray[i][j];
        }
        if((O2.y-O1.y+O3.y>=(int)BIH.height)||(O2.x-
O1.x+O3.x>=(int)BIH.width)) printf(WARNING_NO_SPACE);
        return;
    }
    void mirror(Point O1, Point O2, int type){
        Pixel temp;
        if(type == 1){
             //horizontally
            for(int i=O1.y; i<=O2.y; i++) for(int j=O1.x; j<(O1.x+O2.x)
/2; j++){
                temp = PixelArray[i][j];
                PixelArray[i][j] = PixelArray[i][O1.x+O2.x-j];
                PixelArray[i][O1.x+O2.x-j] = temp;
            }
        } else if(type == 2){
             // verically
            for(int j=O1.x; j<=O2.x; j++) for(int i=O1.y; i<(O1.y+O2.y)
/2; i++){
```

```
                temp = PixelArray[i][j];
                PixelArray[i][j] = PixelArray[O1.y+O2.y-i][j];
                PixelArray[O1.y+O2.y-i][j] = temp;
            }
        }
        return;
    }
    void save(const char* name){
        FILE *f = fopen(name, "wb");
        if(f == NULL){
            printf(ERR_FILE);
            exit(1);
        }
        fwrite(&BFH, sizeof(BMPFileHeader), 1, f);
        fwrite(&BIH, sizeof(BMPInfoHeader), 1, f);
        int padding = BIH.width%4;
        unsigned char* temp;
        if(padding) temp = (unsigned char*)calloc(padding, sizeof(unsig
ned char));
        fseek(f, BFH.offset, SEEK_SET);
        for(int i=0; i<(int)BIH.height; i++){
            fwrite(PixelArray[BIH.height-1-
i], sizeof(Pixel), BIH.width, f);
            if(padding) fwrite(temp, sizeof(unsigned char), padding, f);
        }
        fclose(f);
        printf("[BMP] File successfully saved as %s.\n", name);
        return;
    }

    void help(){
        printf("[BMP] --load|-l <filename>, --save|-s <filename> -
 load or save BMP image.\n");
        printf("[BMP] --info|-i - show information about BMP image.\n");
        printf("[BMP] --circle <x1,y1>|<x2,y2,x3,y3> -
 draw circle centered at x1,y1 or inscribed in square at x2,y2,x3,y3. C
entered circle don't requires radius.\n");
        printf("[BMP] --radius|-r <R> -
 set radius R of centered circle; --thickness|-t <T> -
 set thickness T of circle.\n");
        printf("[BMP] --color|-c <color> - set color of circle; --
fill|-f <color> - set color of filling, optional.\n");
        printf("[BMP] Available colors: red, orange, yellow, lime, gree
n, turquoise, aqua, blue, purple, pink, brown, white, gray, black.\n");
        printf("[BMP] --mirror <x1,y1,x2,y2> -
 mirror part of image at x1,y1,x2,y2.\n");
        printf("[BMP] --axis|-a h|v -
 type of mirror (horizontally or vertically).\n");
        printf("[BMP] --copy <x1,y1,x2,y2,x3,y3> -
 copy part of image from x1,y1,x2,y2 to x3,y3.\n");
        printf("[BMP] --help|h -
 show this message. Use this key standalone.\n\n");
        return;
    }
};

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

```
    BMPImage BMP;
    Parameters parameters;
    int opt;
    int longIndex;
    char* pointCoordsRaw;
    int pointCoords[6] = {0};
    const char opts[] = "l:s:hir:t:c:f:a:";
    struct option longOpts[]={
        {"load", required_argument, NULL, 'l'},
        {"save", required_argument, NULL, 's'},
        {"help", no_argument, NULL, 'h'},
        {"info", no_argument, NULL, 'i'},
        {"radius", required_argument, NULL, 'r'},
        {"thickness", required_argument, NULL, 't'},
        {"color", required_argument, NULL, 'c'},
        {"fill", required_argument, NULL, 'f'},
        {"axis", required_argument, NULL, 'a'},
        {"circle", required_argument, NULL, 1},
        {"mirror", required_argument, NULL, 2},
        {"copy", required_argument, NULL, 3},
        {NULL, 0, NULL, 0}
    };
    opterr = 0;

    if(argc==1){
        printf(ERR_FEW_ARGS);
        return 1;
    }
    while((opt = getopt_long(argc, argv, opts, longOpts, &longIndex))!=
-1){
        switch(opt){
            case 'l':
                if(parameters.loadFileF){
                    printf(ERR_SAME_ARGS);
                    return 1;
                }
                parameters.loadFileF = 1;
                strcpy(parameters.loadFile, optarg);
                break;
            case 's':
                if(parameters.saveFileF){
                    printf(ERR_SAME_ARGS);
                    return 1;
                }
                parameters.saveFileF = 1;
                strcpy(parameters.saveFile, optarg);
                break;
            case 'h':
                BMP.help();
                break;
            case 'i':
                parameters.infoF = 1;
                break;
            case 'r':
                if( ((atoi(optarg)==0)&&(optarg[0]!='0')) || parameters.
circleRadius || (atoi(optarg)<0) ){
                    printf(ERR_WRONG_ARGS);
```

```
                    return 1;
                }
                parameters.circleRadius = atoi(optarg);
                break;
            case 't':
                if( ((atoi(optarg)==0)&&(optarg[0]!='0')) || (atoi(opta
rg)<0) ){
                    printf(ERR_WRONG_ARGS);
                    return 1;
                }
                parameters.circleThickness = atoi(optarg);
                break;
            case 'c':
                if(parameters.circleColorCircleF){
                    printf(ERR_SAME_ARGS);
                    return 1;
                }
                parameters.circleColorCircleF = 1;
                parameters.circleColorCircle = BMP.getColor(optarg);
                break;
            case 'f':
                if(parameters.circleColorFillF){
                    printf(ERR_SAME_ARGS);
                    return 1;
                }
                parameters.circleColorFillF = 1;
                parameters.circleColorFill = BMP.getColor(optarg);
                break;
            case 'a':
                if(parameters.mirrorType){
                    printf(ERR_SAME_ARGS);
                    return 1;
                }
                if(!strcmp(optarg, "h")) parameters.mirrorType = 1;
                else if (!strcmp(optarg, "v")) parameters.mirrorType =
2;
                else{
                    printf(ERR_WRONG_ARGS);
                    return 1;
                }
                break;
            case 1:
                if(parameters.circleF){
                    printf(ERR_SAME_ARGS);
                    return 1;
                }
                parameters.circleF = 1;
                pointCoordsRaw = strtok(optarg, ",");
                for(int i=0; i<2; i++){
                    if( (pointCoordsRaw==NULL) || ((atoi(pointCoordsRaw)
==0)&&(pointCoordsRaw[0]!='0')) ){
                        printf(ERR_WRONG_ARGS);
                        return 1;
                    }
                    pointCoords[i] = atoi(pointCoordsRaw);
                    pointCoordsRaw = strtok(NULL, ",");
                }
```

```
                parameters.circleType = 1;
                if(pointCoordsRaw!=NULL){
                    for(int i=2; i<4; i++){
                        if( (pointCoordsRaw==NULL) || ((atoi(pointCoord
sRaw)==0)&&(pointCoordsRaw[0]!='0')) ){
                            printf(ERR_WRONG_ARGS);
                            return 1;
                        }
                        pointCoords[i] = atoi(pointCoordsRaw);
                        pointCoordsRaw = strtok(NULL, ",");
                    }
                    parameters.circleType = 2;
                }
                if(parameters.circleType==1){
                    parameters.circleO = {pointCoords[0], pointCoords[1]
};
                    if( pointCoordsRaw!=NULL ){
                        printf(ERR_WRONG_ARGS);
                        return 1;
                    }
                }else{
                    parameters.circleA = {pointCoords[0], pointCoords[1]
};
                    parameters.circleB = {pointCoords[2], pointCoords[3]
};
                    if( (pointCoordsRaw!=NULL) || (parameters.circleA.x
 > parameters.circleB.x) || (parameters.circleA.y > parameters.circleB.
y) || parameters.circleRadius ){
                        printf(ERR_WRONG_ARGS);
                        return 1;
                    }
                    if(parameters.circleB.x -
 parameters.circleA.x != parameters.circleB.y - parameters.circleA.y){
                        printf(ERR_WRONG_ARGS);
                        return 1;
                    }
                    parameters.circleRadius = (parameters.circleB.x -
 parameters.circleA.x) / 2;
                    parameters.circleO = {(parameters.circleB.x + param
eters.circleA.x)/2, (parameters.circleB.y + parameters.circleA.y)/2};
                }
                parameters.queueAdd(1);
                break;

            case 2:
                if(parameters.mirrorF){
                    printf(ERR_SAME_ARGS);
                    return 1;
                }
                parameters.mirrorF = 1;
                pointCoordsRaw = strtok(optarg, ",");
                for(int i=0; i<4; i++){
                    if( (pointCoordsRaw==NULL) || ((atoi(pointCoordsRaw)
==0)&&(pointCoordsRaw[0]!='0')) ){
                        printf(ERR_WRONG_ARGS);
                        return 1;
                    }
```

```
                    pointCoords[i] = atoi(pointCoordsRaw);
                    pointCoordsRaw = strtok(NULL, ",");
                }
                parameters.mirrorA = {pointCoords[0], pointCoords[1]};
                parameters.mirrorB = {pointCoords[2], pointCoords[3]};
                if( (pointCoordsRaw!=NULL) || (parameters.mirrorA.x > p
arameters.mirrorB.x) || (parameters.mirrorA.y > parameters.mirrorB.y) )
{
                    printf(ERR_WRONG_ARGS);
                    return 1;
                }
                parameters.queueAdd(2);
                break;

            case 3:
                if(parameters.copyF){
                    printf(ERR_SAME_ARGS);
                    return 1;
                }
                parameters.copyF = 1;
                pointCoordsRaw = strtok(optarg, ",");
                for(int i=0; i<6; i++){
                    if( (pointCoordsRaw==NULL) || ((atoi(pointCoordsRaw)
==0)&&(pointCoordsRaw[0]!='0')) ){
                        printf(ERR_WRONG_ARGS);
                        return 1;
                    }
                    pointCoords[i] = atoi(pointCoordsRaw);
                    pointCoordsRaw = strtok(NULL, ",");
                }
                parameters.copyA = {pointCoords[0], pointCoords[1]};
                parameters.copyB = {pointCoords[2], pointCoords[3]};
                parameters.copyC = {pointCoords[4], pointCoords[5]};
                if( (pointCoordsRaw!=NULL) || (parameters.copyA.x > par
ameters.copyB.x) || (parameters.copyA.y > parameters.copyB.y) ){
                    printf(ERR_WRONG_ARGS);
                    return 1;
                }
                parameters.queueAdd(3);
                break;
            case '?':
                printf(ERR_WRONG_ARGS);
                return 1;
        }
    }
    if(parameters.loadFileF == 0){
        printf(ERR_NO_FILE);
        return 1;
    }
    BMP.load(parameters.loadFile);
    if(parameters.infoF) BMP.info();
    for(int i=0; i<8; i++){
        switch(parameters.queue[i]){
            case 0:
                break;
            case 1:
```

```
                if( (parameters.circleRadius < parameters.circleThickne
ss) || BMP.checkPoint(parameters.circleO) || (parameters.circleThicknes
s == 0) || (parameters.circleRadius == 0) ){
                    printf(ERR_WRONG_ARGS);
                    return 1;
                }
                BMP.drawCircle(parameters.circleO, parameters.circleRad
ius, parameters.circleThickness, parameters.circleColorCircle, paramete
rs.circleColorFillF, parameters.circleColorFill);
                printf("[BMP] Drawing circle in point %d,%d...\n", para
meters.circleO.x, parameters.circleO.y);
                break;
            case 2:
                if( (BMP.checkPoint(parameters.mirrorA)) || (BMP.checkP
oint(parameters.mirrorB)) || (parameters.mirrorType == 0) ){
                    printf(ERR_WRONG_ARGS);
                    return 1;
                }
                BMP.mirror(parameters.mirrorA, parameters.mirrorB, para
meters.mirrorType);
                printf("[BMP] Mirror image inside points %d,%d %d,%d...
\n", parameters.mirrorA.x, parameters.mirrorA.y, parameters.mirrorB.x,
parameters.mirrorB.y);
                break;
            case 3:
                if ( BMP.checkPoint(parameters.copyA) || BMP.checkPoint
(parameters.copyB) || BMP.checkPoint(parameters.copyC) ){
                    printf(ERR_WRONG_ARGS);
                    return 1;
                }
                BMP.copy(parameters.copyA, parameters.copyB, parameters.
copyC);
                printf("[BMP] Copy image inside points %d,%d %d,%d to %
d,%d...\n", parameters.copyA.x, parameters.copyA.y, parameters.copyB.x,
 parameters.copyB.y, parameters.copyC.x, parameters.copyC.y);
                break;
        }
    }
    if( (parameters.saveFileF == 0) && (parameters.queue[0] == 0) ) ret
urn 0;
    if(parameters.saveFileF == 0) printf(WARNING_SAVE);
    BMP.save(parameters.saveFile);
    return 0;
}
```