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. 0382 _____ Литягин С.М.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент: Литягин С.М.

Группа 0382

Тема работы: Обработка BMP файлов

Исходные данные:

Вариант 8

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. Поиск всех залитых прямоугольников заданного цвета. Требуется найти все прямоугольники заданного цвета и обвести их линией. Функционал определяется:
    - Цветом искомых прямоугольников
    - Цветом линии для обводки

- Толщиной линии для обводки

2. 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
3. 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
4. Разделяет изображение на N*M частей. Реализация: либо провести линии заданной толщины, тем самым разделив изображение либо сохранение каждой части в отдельный файл. -- по желанию студента (можно и оба варианта). Функционал определяется:
   - Количество частей по “оси” Y; Количество частей по “оси” X
   - Толщина линии
   - Цвет линии
   - Либо путь куда сохранить кусочки.

Содержание пояснительной записки:

«Содержание» , «Введение» , «Ход выполнения работы» , «Тестирование», «Заключение», «Список использованных источников», «Приложение А. Исходный код программы»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 05.04.2021

Дата сдачи реферата: 25.05.2021

Дата защиты реферата: 27.05.2021

Студент _____ Литягин С.М.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В курсовой работе была разработана программа для обработки BMP-изображения на языке C, которая изменяет изображение в зависимости от команд пользователя.

Программа имеет Command Line Interface (CLI). Также доступен вывод руководства по программе (общий вид команды для работы с программой и информация о доступных ключах).

Функционал программы определяется поиском всех залитых прямоугольников заданного цвета, рисованием окружности, фильтром rgb-компонент, разделением изображения на N*M частей.

При вводе некорректных ключей или неправильных данных пользователю выводятся ошибки.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – создать редактор для BMP файлов на языке C с CLI, производящую обработку изображения согласно командам пользователя.

Чтобы достичь данную цель, нужно решить следующие задачи:

- Написание интерфейса CLI
- Чтение и запись BMP файла
- Создание структур для работы с данными файла
- Реализация заданных условием задачи функций
- Обработка возможных ошибок

1.ЗАДАНИЕ

Вариант 8

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:

5. Поиск всех залитых прямоугольников заданного цвета. Требуется найти все прямоугольники заданного цвета и обвести их линией. Функционал определяется:
    - Цветом искомых прямоугольников
    - Цветом линии для обводки
    - Толщиной линии для обводки
6. 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

7. 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
8. Разделяет изображение на N*M частей. Реализация: либо провести линии заданной толщины, тем самым разделив изображение либо сохранение каждой части в отдельный файл. -- по желанию студента (можно и оба варианта). Функционал определяется:
   - Количество частей по “оси” Y; Количество частей по “оси” X
   - Толщина линии
   - Цвет линии
   - Либо путь куда сохранить кусочки.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Структуры

Всем созданным структурам назначаем их имена с помощью typedef. Создаем структуры **BitmapFileHeader** и **BitmapInfoHeader** с полями, соответствующими версии формата **BMP**, с которым будем работать. Структура **Rgb** определяет цвет пикселя (параметры: красный, зеленый, синий). Структура **BMP** содержит в себе информацию о файле (*BitmapFileHeader* и *BitmapInfoHeader*), а также массив пикселей *arr*.

Все структуры оборачиваем в *#pragma pack(push,1)* и *#pragma pack(pop)*. Это нужно, чтобы размер выравнивания устанавливался в 1 байт (второе выражение это отменяет), иначе размер структур в памяти будет зависеть от компилятора.

2.2. Интерфейс командной строки

Функция *main()* принимает в себя два аргумента: *argc* (количество аргументов) и *argv* (строка с аргументами). С помощью функции *getopt_long()* (подключается библиотекой *getopt.h*) программа получает определенную команду (возможные команды указаны в аргументе функции в следующем виде "*hic:C:1:2:b:f:R:?*"). В соответствие с полученным флагом (поддерживается короткая и длинная запись флага) программа вызывает требуемую пользователем функцию с аргументами, заданными пользователем.

Для работы с программой пользователю требуется ввести в терминал строку следующего вида:

./<исполняемый файл> <имя входного файла> -f/--flag <arg1>,... <имя выходного файла>

Ниже представлена информация о флагах и аргументах:

1. Смена цвета заливки: --colour1/-c <r>,<g>,<b>

2. Смена цвета линии обводки: --colour2/-C <r>,<g>,<b>

3. Нарисовать окружность: --circle1/-1 <вершина по ширине>,<вершина по высоте>,<радиус>,<толщина обводки>,<закрашенность>

параметр закрашенности: если 0 - не закрашивать, если любое другое число - закрашивать

4. Нарисовать окружность: --circle2/-2 <x1>,<y1>,<x2>,<y2>,<толщина обводки>,<закрашенность>

параметр закрашенности: если 0 - не закрашивать, если любое другое число - закрашивать

5. Деление на N*M частей: --cutBMP/-b <x1>,<y1>,<толщина обводки>

6. RGB-фильтр: --filter/-f <value>,<component>

параметр компоненты должен быть или 'r', или 'g', или 'b'

7. Поиск всех прямоугольников: --findRectangle/-R <толщина линии обводки>

8. Информация об изображении: --info/-i

9. Справка: --help/-h

2.3. Чтение файла

Считывание изображение происходит с помощью функции *int openBMP(const char* path, BMP* picture)*. Если файл, переданный как аргумент path, не удается открыть, то выводится ошибка. Иначе – проверяется, оканчивается ли переданная строка на “.bmp”. Если нет – выдается ошибка. Иначе – считывается BitmapInfoHeader и BitmapFileHeader. Если глубина цвета не равно 24 битам на пиксель, или файл имеет таблицу цветов, или файл сжат, то пользователь также получает ошибку. Если же все хорошо, то считывается массив пикселей. Файл закрывается.

2.4. Первая функция

int Circle(BMP picture, float x, float y, float r, int lineWidth, Rgb colourLine, int fill, Rgb colour, char* path);*

Первая функция представляет из себя отрисовку окружности. С помощью формулы уравнения окружности на плоскости проверяется, удовлетворяет ли позиция пикселя данное уравнение. Если да, то пиксель будет раскрашен. Таким образом отрисовывается окружность (см. пример в **Тестировании**).

2.5. Вторая функция

int cutBMP(BMP picture, uint16_t X, uint16_t Y, uint16_t lineWidth, Rgb colour, char* path);*

Нарезка изображения на M*N частей. Внутри функции рассчитываются позиции, где клетка должны быть закрашена, чтобы изображение было корректно поделено. Потом эти пиксели закрашиваются указанным цветом.

2.6. Третья функция

int RGB(BMP picture, char* name, uint8_t value, char* path);*

Фильтр RGB-компонента. Функция меняет значение определенной компоненты пикселей всего изображения.

2.7. Четвертая функция

int findRectangle(BMP picture, Rgb colourR, Rgb colourL, int lineWidth, char* path);*

Функция поиска прямоугольников определенного цвета. Функция проходит по всем пикселям, пока не найдет пиксель нужного оттенка. Если это произошло, то она идет по его контуру, обрисовывая своим шагом прямоугольник. Если удалось обрисовать прямоугольник, грани которого не касаются пикселей такого же оттенка, то функция запоминает его нижний угол и верхний в массив. В конце происходит обрисовка всех найденных прямоугольников дополнительной функцией *int frameRectangle(BMP* picture, int x1, int y1, int x2, int y2, int lineWidth, Rgb colour, char* path);*

2.8. Сохранение файла

Новый bmp-файл сохраняется функцией *int saveBMP(BMP* picture, char* path);*

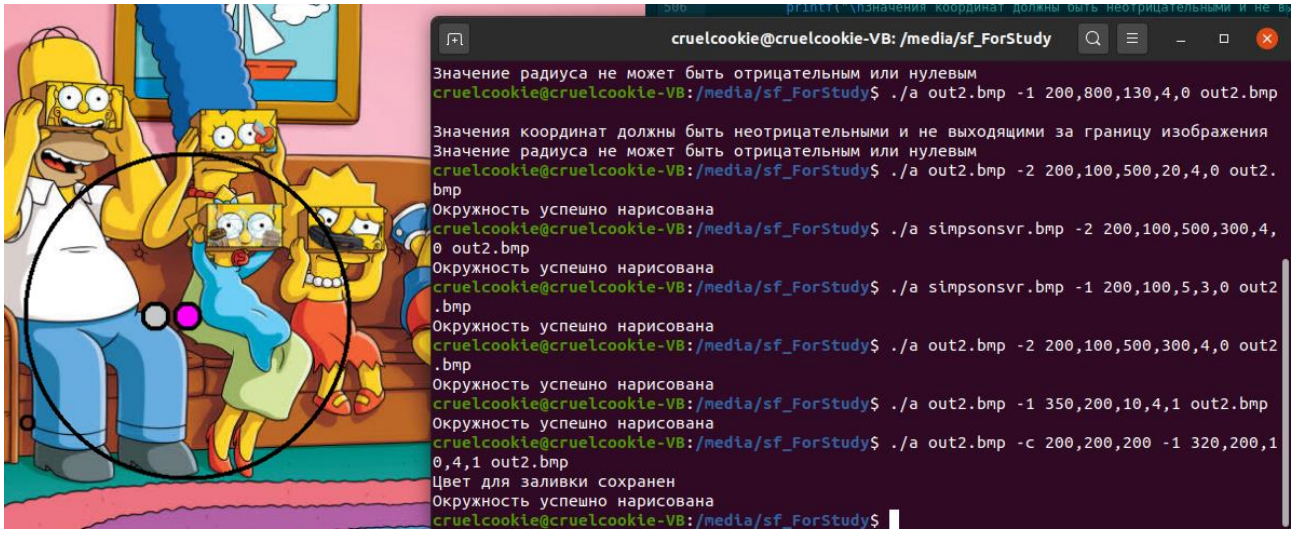
Сначала функция открывает файл с указанным названием *path* (если он не имеет расширение “.bmp”, то выводится соответствующая ошибка еще в начале обработки команды пользователя). В новый файл записываются данные *BitmapFileHeader* и *BitmapInfoHeader*, а затем каждая строка массива пикселей *arr*. Файл закрывается.

3. ТЕСТИРОВАНИЕ

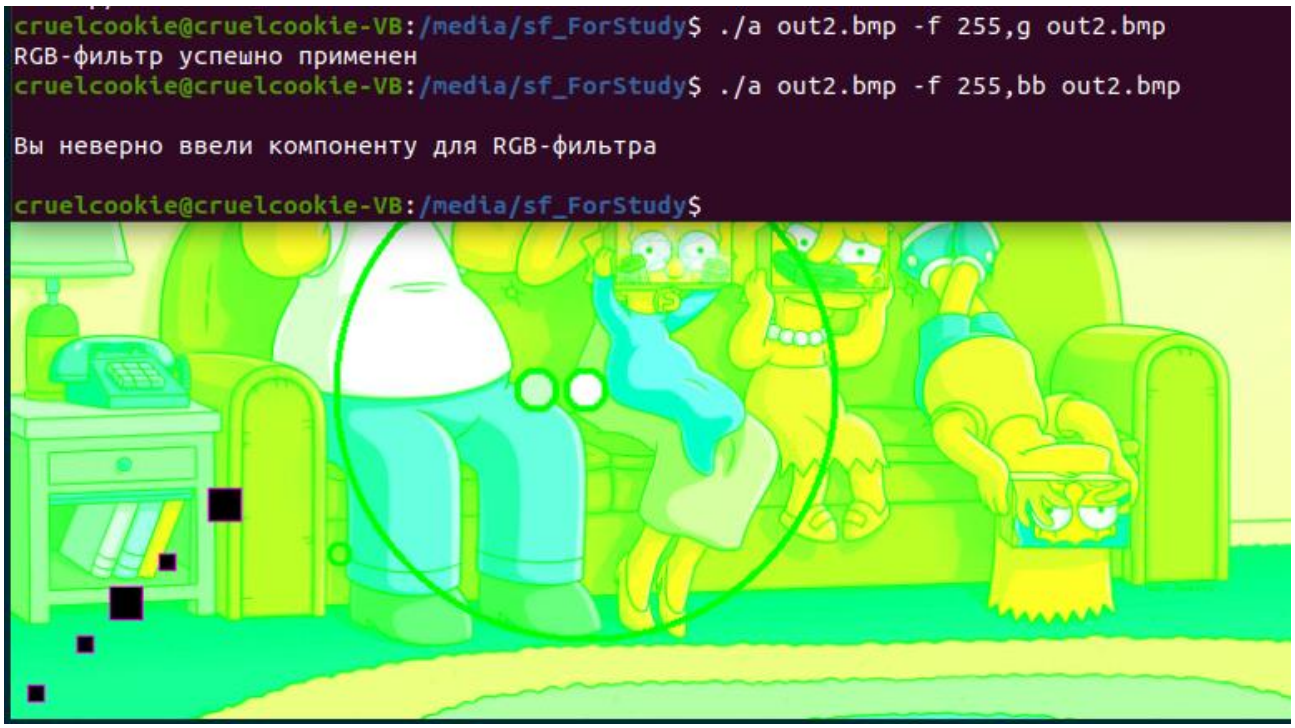
3.1. Вывод справки:

```
cruelcookie@cruelcookie-VB:/media/sf_ForStudy$ ./a -help
                    Справка по эксплуатации:

        Чтобы вызвать помощь, введите в терминал следующее:
                        ./a или ./a -h/--help

        Чтобы выполнить какое-то действие, введите в терминал следующее:
./a <имя входного файла> -f/--flag <arg1>,<arg2>,... <имя выходного файла>
                (на месте многоточия указываются аргументы.)

        Флаги (агрументы флагов следует разделять запятой):

        Системой заданы 2 цвета: цвет заливки (относится и к цвету поиска прямоугольников) и цвет обводки
        Если вы желаете изменить эти цвета перед использованием какой-либо функции, используйте следующие ключи:
         Смена цвета заливки: --colour1/-c <r>,<g>,<b>
         Смена цвета линии обводки: --colour2/-C <r>,<g>,<b>
1.Нарисовать окружность:
--circle1/-1 <вершина по ширине>,<вершина по высоте>,<радиус>,<толщина обводки>,<закрашенность>
        параметр закрашенности: если 0 - не закрашивать, если любое другое число - закрашивать
2.Нарисовать окружность:
--circle2/-2 <x1>,<y1>,<x2>,<y2>,<толщина обводки>,<закрашенность>
        параметр закрашенности: если 0 - не закрашивать, если любое другое число - закрашивать
3.Деление на N*M частей:
--cutBMP/-b <x1>,<y1>,<толщина обводки>
4.RGB-фильтр:
--filter/-f <value>,<component>
        параметр компоненты должен быть или 'r', или 'g', или 'b'
5.Поиск всех прямоугольников:
--findRectangle/-R <толщина линии обводки>
6.Информация об изображении:
--info/-i
7.Справка:
--help/-h
cruelcookie@cruelcookie-VB:/media/sf_ForStudy$ ./a fafaf
                    Справка по эксплуатации:

        Чтобы вызвать помощь, введите в терминал следующее:
                        ./a или ./a -h/--help

        Чтобы выполнить какое-то действие, введите в терминал следующее:
./a <имя входного файла> -f/--flag <arg1>,<arg2>,... <имя выходного файла>
                (на месте многоточия указываются аргументы.)

        Флаги (агрументы флагов следует разделять запятой):

        Системой заданы 2 цвета: цвет заливки (относится и к цвету поиска прямоугольников) и цвет обводки
        Если вы желаете изменить эти цвета перед использованием какой-либо функции, используйте следующие ключи:
         Смена цвета заливки: --colour1/-c <r>,<g>,<b>
         Смена цвета линии обводки: --colour2/-C <r>,<g>,<b>
1.Нарисовать окружность:
--circle1/-1 <вершина по ширине>,<вершина по высоте>,<радиус>,<толщина обводки>,<закрашенность>
        параметр закрашенности: если 0 - не закрашивать, если любое другое число - закрашивать
2.Нарисовать окружность:
--circle2/-2 <x1>,<y1>,<x2>,<y2>,<толщина обводки>,<закрашенность>
        параметр закрашенности: если 0 - не закрашивать, если любое другое число - закрашивать
3.Деление на N*M частей:
--cutBMP/-b <x1>,<y1>,<толщина обводки>
4.RGB-фильтр:
--filter/-f <value>,<component>
        параметр компоненты должен быть или 'r', или 'g', или 'b'
5.Поиск всех прямоугольников:
--findRectangle/-R <толщина линии обводки>
6.Информация об изображении:
--info/-i
7.Справка:
--help/-h
```

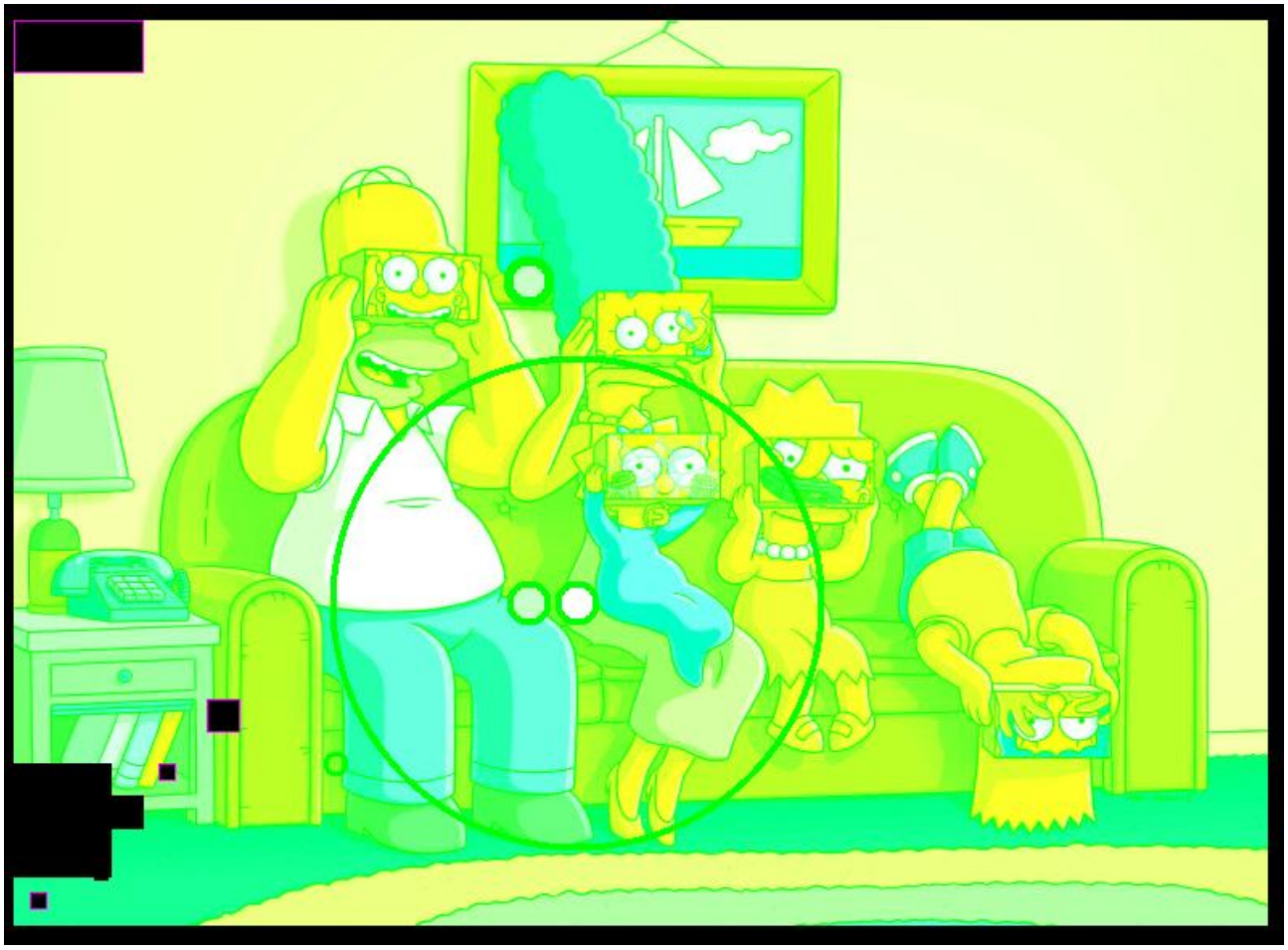
3.2. Рисование окружностей:



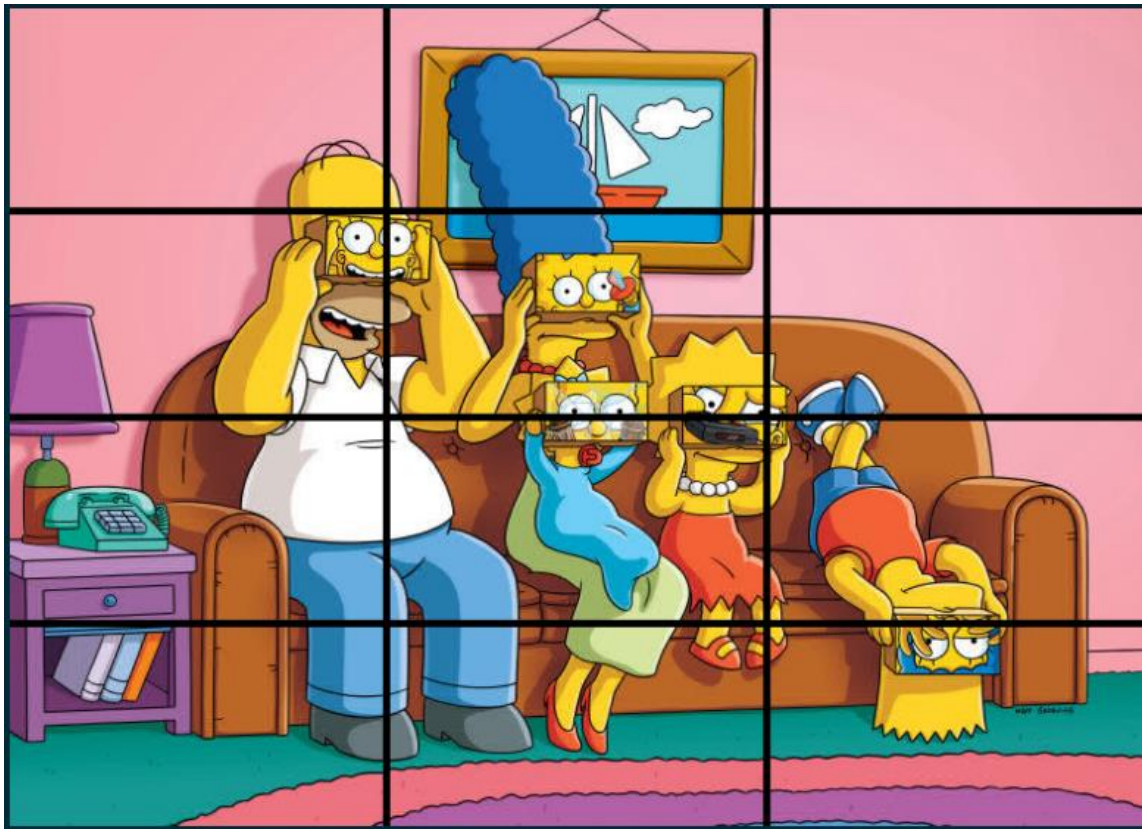
3.3. RGB-фильтр

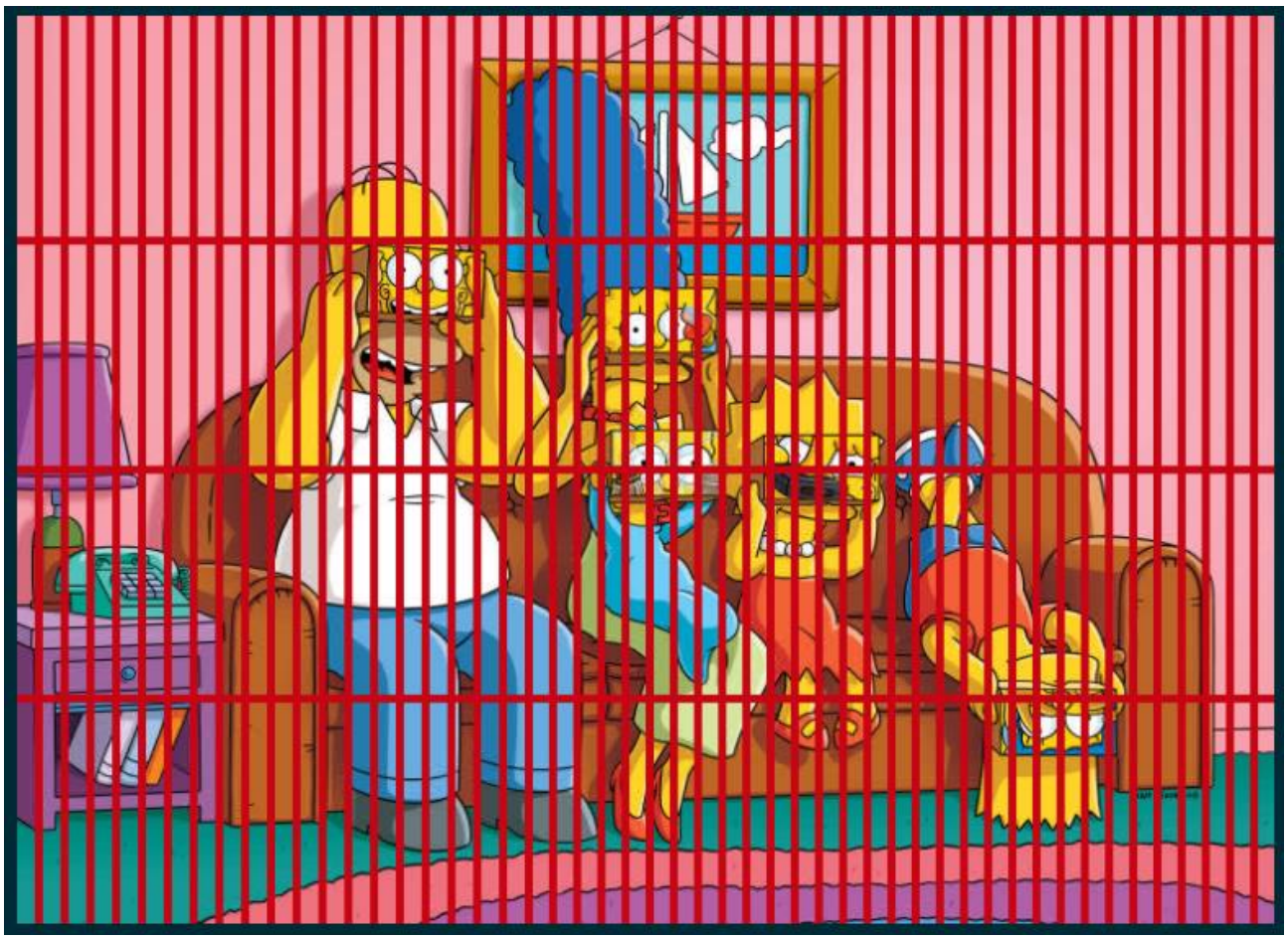


3.4. Поиск прямоугольников



3.4. Разделение изображения





ЗАКЛЮЧЕНИЕ

В ходе работы курсовой работы был написан редактор для BMP-файлов, который имеет несколько функций, поставленных задачей. А именно: рисование кругов двумя способами, поиск прямоугольников заданного цвета, разделение картинки на части, фильтр RGB-компоненты. Также был реализован интерфейс взаимодействия с пользователем с помощью функции библиотеки *getopt.h*.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.Сайт (общая информация): https://ru.wikipedia.org/wiki/БМР

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.c

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

#pragma pack(push,1)
typedef struct{
  uint16_t signature;
  uint32_t filesize;
  uint16_t reserved1;
  uint16_t reserved2;
  uint32_t pixelArrOffset;
} BitmapFileHeader;

typedef struct
{
      unsigned int headerSize;
      unsigned int width;
      unsigned int height;
      unsigned short planes;
      unsigned short bitsPerPixel;
      unsigned int compression;
      unsigned int imageSize;
      unsigned int xPixelsPerMeter;
      unsigned int yPixelsPerMeter;
      unsigned int colorsInColorTable;
      unsigned int importantColorCount;
} BitmapInfoHeader;

typedef struct{
      uint8_t b;
      uint8_t g;
      uint8_t r;
} Rgb;

typedef struct{
  BitmapFileHeader bmfh;
  BitmapInfoHeader bmih;
  Rgb** arr;
} BMP;

#pragma pack(pop)

//-------------------------------Вспомогательные функции-----------------
-------

int max(int x, int y){
  if(x > y){
    return x;
  }
```

```
  return y;
}

int min(int x, int y){
  if(x < y){
    return x;
  }
  return y;
}

//------------------Разные функции с выводом информации------------------
-------

void printInfo(BMP* picture){

  printf("signature:\t%x (%hu)\n", picture->bmfh.signature, picture-
>bmfh.signature);
  printf("filesize:\t%x (%u)\n", picture->bmfh.filesize, picture-
>bmfh.filesize);
  printf("reserved1:\t%x (%hu)\n", picture->bmfh.reserved1, picture-
>bmfh.reserved1);
  printf("reserved2:\t%x (%hu)\n", picture->bmfh.reserved2, picture-
>bmfh.reserved2);
  printf("pixelArrOffset:\t%x (%u)\n", picture->bmfh.pixelArrOffset,
picture->bmfh.pixelArrOffset);

     printf("headerSize:\t%x (%u)\n", picture->bmih.headerSize, picture-
>bmih.headerSize);
     printf("width:     \t%x (%u)\n", picture->bmih.width, picture-
>bmih.width);
     printf("height:    \t%x (%u)\n", picture->bmih.height, picture-
>bmih.height);
     printf("planes:    \t%x (%hu)\n", picture->bmih.planes, picture-
>bmih.planes);
     printf("bitsPerPixel:\t%x (%hu)\n", picture->bmih.bitsPerPixel,
picture->bmih.bitsPerPixel);
     printf("compression:\t%x (%u)\n", picture->bmih.compression,
picture->bmih.compression);
     printf("imageSize:\t%x (%u)\n", picture->bmih.imageSize, picture-
>bmih.imageSize);
     printf("xPixelsPerMeter:\t%x (%u)\n", picture-
>bmih.xPixelsPerMeter, picture->bmih.xPixelsPerMeter);
     printf("yPixelsPerMeter:\t%x (%u)\n", picture-
>bmih.yPixelsPerMeter, picture->bmih.yPixelsPerMeter);
     printf("colorsInColorTable:\t%x (%u)\n", picture-
>bmih.colorsInColorTable, picture->bmih.colorsInColorTable);
     printf("importantColorCount:\t%x (%u)\n", picture-
>bmih.importantColorCount, picture->bmih.importantColorCount);
}

void printHelp(){
  printf("\t\t     \033[4mСправка по эксплуатации:\033[0m\n\n");
  printf("\tЧтобы вызвать помощь, введите в терминал следующее:\n");
  printf("\t\t\t./a или ./a -h/--help\n\n");
  printf("\tЧтобы выполнить какое-то действие, введите в терминал
следующее:\n");
```

```
  printf("./a <имя входного файла> -f/--flag <arg1>,<arg2>,... <имя
выходного файла>\n");
  printf("\t\t(на месте многоточия указываются аргументы.)\n\n");
  printf("\t\033[4mФлаги (агрументы флагов следует разделять
запятой):\033[0m\n\n");

  printf("\tСистемой заданы 2 цвета: цвет заливки (относится и к цвету
поиска прямоугольников) и цвет обводки\n");
  printf("\tЕсли вы желаете изменить эти цвета перед использованием
какой-либо функции, используйте следующие ключи:\n");
  printf("\t Смена цвета заливки: --colour1/-c <r>,<g>,<b>\n");
  printf("\t Смена цвета линии обводки: --colour2/-C <r>,<g>,<b>\n");

  printf("1.Нарисовать окружность: \n--circle1/-1 <вершина по
ширине>,<вершина по высоте>,<радиус>,<толщина
обводки>,<закрашенность>\n");
  printf("\tпараметр закрашенности: если 0 - не закрашивать, если любое
другое число - закрашивать\n");
  printf("2.Нарисовать окружность: \n--circle2/-2
<x1>,<y1>,<x2>,<y2>,<толщина обводки>,<закрашенность>\n");
  printf("\tпараметр закрашенности: если 0 - не закрашивать, если любое
другое число - закрашивать\n");
  printf("3.Деление на N*M частей: \n--cutBMP/-b <x1>,<y1>,<толщина
обводки>\n");
  printf("4.RGB-фильтр: \n--filter/-f <value>,<component>\n");
  printf("\tпараметр компоненты должен быть или 'r', или 'g', или
'b'\n");
  printf("5.Поиск всех прямоугольников: \n--findRectangle/-R <толщина
линии обводки>\n");
  printf("6.Информация об изображении: \n--info/-i\n");
  printf("7.Справка: \n--help/-h\n");
}

//-------------------------Проверка на .bmp-----------------------------
-------

int checkBMP(const char* path){
  if(strlen(path) <= 4) return 0;
  if(path[strlen(path)-1] != 'p' || path[strlen(path)-2] != 'm'
           || path[strlen(path)-3] != 'b' || path[strlen(path)-4] !=
'.') return 0;
  return 1;
}

//-------------------------Открытие BMP файла---------------------------
-------

int openBMP(const char* path, BMP* picture){
  FILE* f = fopen(path, "rb");
  if(!f){
    printf("\nИсходный файл не найден\n");
    return 0;
  }
  if(!checkBMP(path)){
    printf("\nВведен неверный формат исходного файла\n");
    return 0;
  }
```

```
  fread(&(picture -> bmfh), 1, sizeof(BitmapFileHeader), f);
  fread(&(picture -> bmih), 1, sizeof(BitmapInfoHeader), f);

     if(picture -> bmih.bitsPerPixel != 24){
    printf("\nИзображение данной глубины цвета не поддерживается\n");
    printf("Глубина должна быть 24 бита\n");
    return 0;
  }
  if(picture -> bmih.colorsInColorTable != 0){
          printf("Таблица цветов изображения не поддерживается\n");
    return 0;
  }
  if(picture -> bmih.compression != 0){
    printf("Сжатые изображения не поддерживаются\n");
    return 0;
  }

  picture->arr = malloc(picture -> bmih.height*sizeof(Rgb*));
  for(int i = 0; i < picture -> bmih.height; i++){
    picture->arr[i] = malloc(picture -> bmih.width*sizeof(Rgb) + (picture
-> bmih.width * 3)%4);
    fread(picture->arr[i], 1, picture -> bmih.width*sizeof(Rgb)+(picture
-> bmih.width * 3)%4, f);
  }
  fclose(f);
  return 1;
}

//--------------------------Сохранение-----------------------------------
-------

int saveBMP(BMP* picture, char* path){

  uint32_t W = picture -> bmih.width;
  uint32_t H = picture -> bmih.height;
  if(!checkBMP(path)){
    printf("\nВведен неверный формат конечного файла\n");
    return 0;
  }
  FILE* f = fopen(path, "wb");
  fwrite(&(picture->bmfh), 1, sizeof(BitmapFileHeader), f);
  fwrite(&(picture->bmih), 1, sizeof(BitmapInfoHeader), f);
  uint32_t w = W*sizeof(Rgb) + (W*3)%4;
  for(int i = 0; i < H; i++){
    fwrite(picture -> arr[i], 1, w, f);
  }
  fclose(f);
  return 1;
}

//----------------------Нарезка картинка N*M---------------------------
-------

int cutBMP(BMP* picture, uint16_t X, uint16_t Y, uint16_t lineWidth, Rgb
colour, char* path){
  uint32_t H = picture->bmih.height;
```

```
  uint32_t W = picture->bmih.width;
  uint16_t XpixelPerPart = (W-lineWidth*X)/(X+1);
  uint16_t YpixelPerPart = (H-lineWidth*Y)/(Y+1);
  uint16_t XoverPixels = W - XpixelPerPart*(X+1) - X*lineWidth;
  uint16_t YoverPixels = H - YpixelPerPart*(Y+1) - Y*lineWidth;
  uint16_t koordX = 0;
  uint16_t koordY = 0;

  for(int i = 0; i < H; i++){
    uint16_t p = XoverPixels;
    for(int j = 0; j < X; j++){
      koordX += XpixelPerPart;
      /*if(p > XpixelPerPart){
        koordX += p/XpixelPerPart;
        p -= p/XpixelPerPart;
      }*/
      if(p > 0){
        koordX += 1;
        p--;
      }
      for(int k = 0; k < lineWidth; k++){
        if(koordX <= W){
          koordX += 1;
          picture->arr[i][koordX-1] = colour;
        }
      }
    }
    koordX = 0;
  }
  for(int i = 0; i < W; i++){
    uint16_t u = YoverPixels;
    for(int j = 0; j < Y; j++){
      koordY += YpixelPerPart;
      /*if(u > YpixelPerPart){
        koordY += u/YpixelPerPart;
        u -= u/YpixelPerPart;
      }*/
      if(u > 0){
        koordY += 1;
        u--;
      }
      for(int k = 0; k < lineWidth; k++){
        if(koordY <= H){
          koordY += 1;
          picture->arr[koordY-1][i] = colour;
        }
      }
    }
    koordY = 0;
  }
  saveBMP(picture, path);
  return 1;
}

//-----------------------RGB фильтр-------------------------------------
-------
int RGB(BMP* picture, char* name, uint8_t value, char* path){
```

```
  uint32_t W = picture -> bmih.width;
  uint32_t H = picture -> bmih.height;

  if(strcmp(name,"r") && strcmp(name,"g") && strcmp(name,"b")){
     printf("\nВы неверно ввели компоненту для RGB-фильтра\n\n");
     return 0;
  }

  for(int i = 0; i < H; i++){
    for(int j = 0; j < W; j++){
      if(!strcmp(name,"r")) picture -> arr[i][j].r = value;
      if(!strcmp(name,"g")) picture -> arr[i][j].g = value;
      if(!strcmp(name,"b")) picture -> arr[i][j].b = value;
    }
  }
  saveBMP(picture, path);
  return 1;
}

//----------------------Новый цвет-------------------------------------
-------

int createColour(Rgb* colour, uint8_t r, uint8_t g, uint8_t b){
  colour->r = r;
  colour->g = g;
  colour->b = b;
  return 1;
}

//--------------------------------Рисование круга--------------------------
-------

int Circle(BMP* picture, float x, float y, float r, int lineWidth, Rgb
colourLine, int fill, Rgb colour, char* path){

  int W = picture -> bmih.width;
  int H = picture -> bmih.height;

  for(int i = 0; i < H; i++){
    for(int j = 0; j < W; j++){
      if((i-y)*(i-y) + (j-x)*(j-x) >= r*r && (i-y)*(i-y) + (j-x)*(j-x) <
(r+lineWidth)*(r+lineWidth)){
        picture -> arr[i][j] = colourLine;
      }
    }
  }
  if(fill){
    for(int i = 0; i < H; i++){
      for(int j = 0; j < W; j++){
        if((i-y)*(i-y) + (j-x)*(j-x) < r*r){
          picture -> arr[i][j] = colour;
        }
      }
    }
  }
  if(!saveBMP(picture, path)) return 0;
  return 1;
```

```
}

//-------------------------------------------------------------------------
-------

int frameRectangle(BMP* picture, int x1, int y1, int x2, int y2, int
lineWidth, Rgb colour, char* path){

    int W = picture -> bmih.width;
    int H = picture -> bmih.height;
    int minY = min(y1, y2);
    int maxY = max(y1, y2);
    int minX = min(x1, x2);
    int maxX = max(x1, x2);
    if(x1 < 0 || x1 >= H || x2 < 0 || x2 >= H || y1 < 0 || y1 >= W || y2
< 0 || y2 >= W){
      printf("\nУказанные значения выходят за пределы изображения\n");
      return 1;
    }
    for(int j = minX - lineWidth + 1; j < maxX + lineWidth; j++){
      if((j >= minX - lineWidth && j <= minX) || (j >= maxX && j <= maxX
+ lineWidth)){
        for(int i = minY - lineWidth + 1; i < maxY + lineWidth; i++){
          if(i >= 0 && i < H && j >= 0 && j < W)picture -> arr[i][j] =
colour;
        }
      }
      for(int i = minY - lineWidth + 1; i < maxY + lineWidth; i++){
        if((i >= minY - lineWidth && i <= minY) || (i >= maxY && i <=
maxY + lineWidth)){
          for(int j = minX - lineWidth + 1; j < maxX + lineWidth; j++){
            if(i >= 0 && i < H && j >= 0 && j < W)picture -> arr[i][j] =
colour;
          }
        }
      }
    }
    saveBMP(picture, path);
    return 0;
}

int findRectangle(BMP* picture, Rgb colourR, Rgb colourL, int lineWidth,
char* path){
  int value;
  int* coords = malloc(value*sizeof(int));
  int width1 = -1;
  int height1 = -1;
  int x1 = -1, y1 = -1;
  Rgb** arr = picture->arr;
  int W = picture -> bmih.width;
  int H = picture -> bmih.height;
  int index1 = 0;
  int index2 = 0;
  int flag = 0;
  for(int i = 0; i < H-2; i++){
    for(int j = 0; j < W-2; j++){
```

```
        if((i == 0 || memcmp(&arr[i-1][j], &colourR, sizeof(Rgb)))&&
!memcmp(&arr[i][j], &colourR, sizeof(Rgb))){
          width1 = j;
          x1 = j;
          while((width1 == W-1 || !memcmp(&arr[i][width1+1], &colourR,
sizeof(Rgb))) && width1+1!=W && (i == 0 || memcmp(&arr[i-1][width1],
&colourR, sizeof(Rgb)))){
            width1++;
          }
          if(i > 0 && width1+1 < W && !memcmp(&arr[i-1][width1],
&colourR, sizeof(Rgb))){
            width1 = -1;
            x1 = -1;
            break;
          }
          if((width1 == W-1 || (i+1 < H && memcmp(&arr[i+1][width1+1],
&colourR, sizeof(Rgb)))) && !memcmp(&arr[i][width1], &colourR,
sizeof(Rgb))){
            height1 = i;
            y1 = i;
            while((height1 == H-1 || !memcmp(&arr[height1+1][width1],
&colourR, sizeof(Rgb))) && height1+1!=H && (width1 == W-1 ||
memcmp(&arr[height1][width1+1], &colourR, sizeof(Rgb)))){
              height1++;
            }
            if(height1+1 < H && width1+1 < W &&
!memcmp(&arr[height1][width1+1], &colourR, sizeof(Rgb))){
              height1 = -1;
              y1 = -1;
              break;
            }
            for(int k = y1; k <= height1; k++){
              if((x1 == 0 || memcmp(&arr[k][x1-1], &colourR,
sizeof(Rgb))) && !memcmp(&arr[k][x1], &colourR, sizeof(Rgb))) flag++;
            }
            if(flag - 1 == height1 - y1 && height1 - y1 >= 2){
              flag = 0;
              for(int l = x1; l <= width1; l++){
                if((height1 == H-1 || memcmp(&arr[height1+1][l],
&colourR, sizeof(Rgb))) && !memcmp(&arr[height1][l], &colourR,
sizeof(Rgb))) flag++;
              }
              if(flag -1 == width1 - x1 && width1 - x1 >=2){
                  for(int x = x1; x <= width1; x++){
                    for(int y = y1; y <= height1; y++){
                      if(!memcmp(&arr[y][x], &colourR, sizeof(Rgb))){
                        index1++;
                      }
                    }
                  }
                  if(index1 == (height1 - y1+1)*(width1 - x1+1)){
                    coords[index2*4 + 0] = x1;
                    coords[index2*4 + 1] = y1;
                    coords[index2*4 + 2] = width1;
                    coords[index2*4 + 3] = height1;
                    index2++;
                    if(index2 == value-1){
```

```
                    value+=20;
                    coords = realloc(coords, value*sizeof(int));
                  }
                }
              }
            }
          }
        }
        x1 = -1;
        y1 = -1;
        height1 = -1;
        width1 = -1;
        flag = 0;
        index1 = 0;
    }
  }
  for(int i = 0; i < index2; i++){
    frameRectangle(picture, coords[i*4 + 0], coords[i*4 + 1], coords[i*4
+ 2], coords[i*4 + 3], lineWidth, colourL, path);
  }
  free(coords);
  saveBMP(picture, path);
  return 1;
}
//------------------------Основная функция------------------------------
-------
int main(int argc, char* argv[]){
  setlocale(LC_ALL, "");

  BMP picture;
  char* path1 = malloc(256*sizeof(char));
  char* path2= malloc(256*sizeof(char));
  int x1 = -1, y1 = -1, x2 = -1, y2 = -1;
  int rad = -1;
  int r = -1, g = -1, b = -1;
  int fill = -1;
  int lineWidth = -1;
  int H,W;
  int value = -1;
  char* componenta = malloc(256*sizeof(char));

  Rgb colour1 = {0, 0, 0};
  Rgb colour2 = {255, 0, 255};

  struct option opts[] = {
    {"help", no_argument, NULL,'h'},
    {"circle1", required_argument, NULL, '1'},
    {"circle2", required_argument, NULL, '2'},
    {"colour1", required_argument, NULL, 'c'},
    {"colour2", required_argument, NULL, 'C'},
    {"cutBMP", required_argument, NULL, 'b'},
    {"filter", required_argument, NULL, 'f'},
    {"info", no_argument, NULL, 'i'},
    {"findRectangle", required_argument, NULL, 'R'}
  };

  if(argc <= 3){
```

```
    printHelp();
    return 0;
  }
  if(argc >= 4){
    int opt;
    int longIndex;
    strcpy(path1,argv[1]);
    strcpy(path2,argv[argc-1]);
    if(!openBMP(path1,&picture)) return 0;
    if(!saveBMP(&picture,path2)) return 0;
    H = picture.bmih.height;
    W = picture.bmih.width;
    opt = getopt_long(argc, argv, "hic:C:1:2:b:f:R:?", opts, &longIndex);
    if(opt == -1){
      printHelp();
      return 0;
    }
    while(opt != -1){
      switch(opt){
        case '1':
            if(sscanf(optarg,"%d,%d,%d,%d,%d\n", &x1, &y1, &rad,
&lineWidth, &fill)){
              if(x1 < 0 || y1 < 0 ||x1 >= W ||y1 >= H || rad <= 0){
                printf("\nЗначения координат должны быть неотрицательными
и не выходящими за границу изображения\n");
                printf("Значение радиуса не может быть отрицательным или
нулевым\n");
                return 0;
              }
              if(lineWidth < 0){
                printf("\nШирина линии обводки не может быть
отрицательной\n");
                return 0;
              }
              if(Circle(&picture, x1, y1, rad, lineWidth, colour1, fill,
colour2, path2)){
                printf("Окружность успешно нарисована\n");
              }
              else{
                return 0;
              }
            }
            else{
              printf("\nВведены не все данные или не в том порядке\n");
              return 0;
            }
            break;
        case '2':
            if(sscanf(optarg,"%d,%d,%d,%d,%d,%d\n", &x1, &y1, &x2, &y2,
&lineWidth, &fill)){
              if(x1 < 0 || x2 < 0 || y1 < 0 || y2 < 0 ||x1 >= W || x2 >=
W || y1 >= H || y2 >= H ){
                printf("\nВсе значения координат должны быть
неотрицательными и не выходящими за границу изображения\n");
                return 0;
              }
              if(max(x1,x2) - min(x1,x2) != max(y1,y2) - min(y1,y2)){
```

```
                printf("Вы ввели не координаты квадрата, попробуйте еще
раз\n");
                return 0;
              }
              if(lineWidth < 0){
                printf("\nШирина линии обводки не может быть
отрицательной\n");
                return 0;
              }
              if(Circle(&picture, (min(x1,x2)+(max(x1,x2) -
min(x1,x2))/2),(min(y1,y2)+(max(y1,y2) - min(y1,y2))/2), (max(x1,x2) -
min(x1,x2))/2, lineWidth, colour1, fill, colour2, path2)){
                printf("Окружность успешно нарисована\n");
              }
              else{
                return 0;
              }
            }
            else{
              printf("\nВведены не все данные или не в том порядке\n");
              return 0;
            }
              break;
        case 'c':
            if(sscanf(optarg,"%d,%d,%d\n", &r, &g, &b)){
              if(r < 0 || r > 255 || g < 0 || g > 255 || b < 0 || b >
255){
                printf("\nВведены неверные значения для нового цвета
заливки\n");
                printf("Все значения должны быть в диапозоне от 0 до
255\n");
                return 0;
              }
              if(createColour(&colour2, r, g, b)){
                printf("Цвет для заливки сохранен\n");
              }
              else{
                return 0;
              }
            }
            else{
              printf("\nВведены не все данные или не в том порядке\n");
              return 0;
            }
              break;
        case 'C':
            if(sscanf(optarg,"%d,%d,%d\n", &r, &g, &b)){
              if(r < 0 || r > 255 || g < 0 || g > 255 || b < 0 || b >
255){
                printf("\nВведены неверные значения для нового цвета
контура\n");
                printf("Все значения должны быть в диапозоне от 0 до
255\n");
                return 0;
              }
              if(createColour(&colour1, r, g, b)){
                printf("Цвет для обводки сохранен\n");
```

```
            }
            else{
              return 0;
            }
          }
          else{
            printf("\nВведены не все данные или не в том порядке\n");
            return 0;
          }
            break;
        case 'b':
          if(sscanf(optarg,"%d,%d,%d\n", &x1, &y1, &lineWidth)){
            if(x1 < 0, y1 < 0){
              printf("\nКоличество линий не может быть меньше нуля\n");
              return 0;
            }
            if(W - lineWidth*x1 < x1+1 || H - lineWidth*y1 < y1+1){
              printf("\nЗадайте меньшую толщину линии-разделителя\n");
              return 0;
            }
            if(cutBMP(&picture, x1, y1, lineWidth, colour1, path2)){
              printf("\nКартинка успешно разделена\n");
            }
            else{
              return 0;
            }
          }
          else{
            printf("\nВведены не все данные или не в том порядке\n");
            return 0;
          }
          break;
      case 'i':
          printInfo(&picture);
          break;
      case 'f':
          if(sscanf(optarg,"%d,%s\n", &value, componenta)){
            if(value >= 256 || value < 0){
              printf("\nВы неверно ввели значение для компоненты RGB-
фильтра\n\n");
              return 0;
            }
            if(RGB(&picture,componenta,value,path2)){
              printf("RGB-фильтр успешно применен\n");
            }
            else{
              return 0;
            }
          }
          else{
            printf("\nВведены не все данные или не в том порядке\n");
            return 0;
          }
          break;
      case 'R':
          if(sscanf(optarg,"%d\n", &lineWidth)){
            if(lineWidth < 0){
```

```
              printf("\nТолщина линии обводки не может быть
отрицательной\n\n");
              return 0;
            }
            if(findRectangle(&picture, colour1, colour2, lineWidth,
path2)){
              printf("Прямоугольники найдены\n");
            }
            else{
              return 0;
            }
          }
          else{
            printf("\nВведены не все данные или не в том порядке\n");
            return 0;
          }
          break;
      case 'h':
      case '?':
      default:
          printHelp();
          return 0;
      }
      opt = getopt_long(argc, argv, "hic:C:1:2:b:f:R:?", opts,
&longIndex);
    }
  }
  saveBMP(&picture, path2);
  return 0;
}
```