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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений на языке С

Студент гр. 1384 _____ Камынин А. А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2022

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент: Камынин А. А.

Группа 1384

Тема работы : Обработка изображений на языке C

Исходные данные:
На вход программе подается bmp-файл, в котором: 24 бита на цвет, без сжатия, файл всегда сооветствует формату BMP. Программа должна иметь CLI или GUI.

Содержание пояснительной записки:
Аннотация, Содержание, Введение, Отчет, Примеры работы программы, Заключение

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 22.03.2022

Дата сдачи реферата: 01.06.2022

Дата защиты реферата: 01.06.2022

Студент _____ Камынин А. А.

Преподаватель _____ Жангиров Т. Р.

АННОТАЦИЯ

Программа на CLI должна быть реализована подобно Linux-подобным утилитам. В программе важно наличие справки, которая распечатывается при вызове утилиты без аргументов или стандартными ключами; обработка всех исключительных случаев; для каждого инструмента должны быть соответствующие ключи и ключи для его конфигурирования, при этом для всех (для которых это имеет смысл) должны быть как полные, так и сокращенные версии.

Программа, управляемая аргументами командной строки, позволяет изменять изображение формата BMP, а именно: поиск всех залитых прямоугольников заданного цвета, рисование окружности, фильтр rgb-компонента, разделение изображения на N*M частей.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках работы необходимо реализовать программу по обработке BMP-изображений. Программа должна иметь CLI или GUI, и реализовывать следующий функционал:

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 также может быть залитой или нет, и если пользователь выбирает первый вариант, то определяется и цвет заливки.
3. Фильтр rgb-компонент. Этот инструмент должен позволять для всего изображения либо установить в 0 либо установить в 255 значение заданной компоненты. Функционал определяется: компонентой, которую требуется изменить, значение, в какое 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. Функционал определяется: количеством частей по «оси» Y, количество частей по «оси» X, толщиной линии, цветом линии, либо путем, в который нужно сохранить кусочки.

1. ПОДГОТОВКА К ОБРАБОТКЕ ИЗОБРАЖЕНИЙ

1.1. Считывание аргументов и опций. Функция main().

Перед считыванием необходимых данных для обработки изображения, инициализируются переменные и структуры, необходимые для дальнейшей работы функций. Перед работой функции getopt_long(), осуществляется проверка числа аргументов, переданных функции main - если этот аргумент один, то переменной err, в которой хранится значение ошибки, присваивается KEY_ERROR. Дальше в цикле while() при помощи оператора switch перебираются переданные параметры со значениями для обработки изображения.

Ключевым параметром является “--open”, или соответсвующий ему короткий ключ, “-o”. Значение, на которое указывает optarg с помощью функции strcat() копируется в строку name_file, которая хранит название (путь) изображения, к которому необходимо применить изменения. Далее считывается массив пикселей Rgb** arr, с помощью функции ReadFile(). Флажку write_flag присваивается значение 1, которое свидетельствует о том, что файл был считан и можно с ним работать.

Для опции “--circle” или “-c” прежде всего осуществляется проверка на наличие обрабатываемого файла. Так, если флаг write_flag имеет значение 0, то изображение не было считано; ошибка INFORMATION_ERROR записывается в переменную err. Снова вызывается функция getopt_long(), которая должна получить следующую опцию: “-r” (“--radius”) или “-q” (“--square”), в зависимости от которой будут различаться входные аргументы, по которым будет рисоваться окружность. В начале с помощью функции ArgChecks() проверяется, не передан ли в качестве аргумента какой-либо ключ. Если это не так, то в переменную idx сохраняется индекс optind. В дальнейшем присваются все необходимые параметры из массива параметров argv[] для рисования окружности, при этом при очередном присваивании проверяется, что ранее не было в переменной err ошибки, что предыдущий аргумент не совпадает с

названием ключа, и что введено верное число параметров. Для аргумента, который характеризует цвет, вызывается функция ChooseColor(), в которой осуществляется проверка корректности введенного цвета. Если по всех считываний параметров не было ошибки, то флажку circle_radius_flag присваивается 1. В ином случае для каждого из условий проверок аргументов в блоке else переменной err присвоится значение ошибки ARG_ERROR. В том случае, если после опции -c не была передана ни опция -r, ни опция -q, в err записывается значение ошибки KEY_ERROR. Так как опция -f является не обязательной для рисования окружности, то в случае отсутствия ошибок осуществляется проверка: если очередное значение функции getopt_long совпадает с ключом -f, то в указатель на название цвета color записывается значение optarg, вызывается функция ChooseColor(), и в случае отсутствия ошибок флажку fill_flag присваивается 1. Дальше, в зависимости от того, какой из флагов, circle_radius_flag или circle_square_flag стоит в 1, вызывается соответствующая функция проверки аргументов, и в случае отсутствия ошибок, сама функция. Значения флажков сбрасываются.

Так как логика считывания и подготовки к вызову функций, обрабатывающих изображения, почти для всех опций одинакова, то удобно ее представить в виде следующей таблицы:

Ключ	Список названия переменных и указателей, в которые записываются входные данные	Функция, проверяющая корректность значений введенных данных	Функция, обрабатывающая изображение
--circle или -c (имеет три зависимых	-	-	-

ключа, которые считываются внутри конструкции switch)			
--radius или -r (для рисования окружности по радиусу)	int radius (радиус окружности), int x1 (координата центра по высоте изображения), int y1 (координата центра по ширине изображения), int thickness (толщина линии окружности), Rgb line_color (цвет линии окружности)	CheckArgCircleRadius(), ChooseColor().	DrawCircle()
--square или -q (рисование окружности по описанному квадрату)	int x1 (координата левого-верхнего угла квадрата по высоте), int	CheckArgCircleSquare(), ChooseColor()	pre_DrawCircle( )

	x2 (координата правого-нижнего угла квадрата по высоте), int y1 (координата левого-верхнего угла квадрата по ширине), int y2 (координата правого-нижнего угла квадрата по ширине), int thickness (толщина линии окружности), Rgb line_color (цвет линии окружности)		
--fill или -f (залитая/не залитая окружность)	int fill_flag (характеризует необходимость заливать/не заливать окружность), Rgb fill_color (цвет заливки окружности)	ChooseColor()	Параметры являются одним из значений аргументов, передаваемых в функции pre_DrawCircle() или DrawCircle().

--filter или -p	char* component (компонент RGB, который требуется изменить), int value (значение, в которое нужно изменить компонент)	CheckArgRgbFilter()	rgb_filter()
--divide или -d	int N (количество кусочков вдоль высоты), int M (количество кусочков вдоль ширины), int thickness (толщина разделяющей линии), char* color (цвет разделяющей линии)	CheckArgDivide(), ChooseColor()	Divide()
--findAllRectangle или -a	Rgb fill_color (цвет заливки прямоугольника), Rgb line_color (цвет линии обводки), int	CheckArgFindAllRectangle(), ChooseColor()	findAllRectangle()

	thickness (толщина линии обводки)		

Опции “--information” (“-i”), “-?”, “-h” не влияют на работу функций, обрабатывающих изображение, однако позволяют вывести информацию об файле (функции printFileHeader() и printInfoHeader()), обозначить ошибку неправильного ключа KEY_ERROR и вывести справку по работе с программой. Case ‘f’, case ‘r’, case ‘q’ прописаны по той причине, что они являются как бы зависимыми ключами, которые вызываются внутри другого, поэтому, если в строке они идут раньше, чем их главный ключ -c, то в переменную err записывается ошибка FLAG_BEFORE_ERROR (также это будет сделано в том случае, если эти ключи указать без ключа -c).

По завершении работы цикла, при отсутствии ошибок, вызывается функция WriteFile(), по умолчанию выводящая результат в файл out.bmp. В случае, если по ходу пробега по параметрам аргументов главной функции или проверки значений в переменной err было записано значение ошибки, объявленной директивой define, выводится справка по работе с программой (вызывается функция PrintHelp()) и выводится текст ошибки.

1.2. Считывание изображения. Функция ReadFile().

На вход функции подается указатель на путь считываемого изображения name_file, указатель на структуры, содержащих заголовок и информацию по файлу BMP bmfh и bmif, указатель на переменную с ошибками err. Инициализируется переменная r, хранящая сведения об ошибке считывания (ининициализируется значением NO_ERROR, т.е. 0), указатель на файл f и двумерный массив arr для пикселей. Если файл был найден, то считываются заголовок и информация о файле. Дальше проверяется формат файла, а именно количество используемых цветов и количество бит на пиксель, которые хранятся в структуре BitmapInfoHeader - в случае неподходящих значений

полей в err записывается ошибка BMP_ERROR. Далее считываются мусорные данные, очищается память. С учетом выравнивания считывается двумерный массив пикселей arr, который представляет собой массив структур Rgb с компонентами b, g, r. Возвращается указатель на массив пикселей arr, в переменную, к который был передан адрес err записывается значение r.

1.3. Вывод изображения. Функция WriteFile().

На вход подаются указатели на структуры BitmapFileHeader и BitmapInfoHeader (bmfh и bmif соответственно), массив пикселей arr. Открывается файл out.bmp на чтение, записываются информации о заголовке и самом файле, с учетом выравнивания записывается массив arr.

1.4. Вывод справки. Функция PrintHelp().

Функция выводит справку, в которой прописаны все ключи и необходимые параметры для работы с функцией. Справка выводится в случае получения ошибки в ходе программы или в случае вызова специальной опции --help (-h).

1.5. Обработка ошибок.

С помощью директивы define определены и присвоены значения следующим видам ошибок:

Название ошибки	Числовое значение	Информация об ошибки, записанная в массиве error_msg
NO_ERROR	0	Ошибок нет
ARG_ERROR	1	Неверно введены данные
OPEN_IMAGE_ERROR	2	Изображение не найдено

KEY_ERROR	3	Неверно введен ключ или его аргументы отсутствуют
COLOR_ERROR	4	Неизвестный цвет
CIRCLE_RADIUS_ERROR	5	По вашим данным невозможно построить круг. Перепроверьте входные параметры
CIRCLE_SQUARE_ERROR	6	Неверно введены координаты квадрата
RGB_FILTER_ERROR	7	Неверные данные для Rgb-фильтра
INFORMATION_ERROR	8	Файл отсутствует или указан позже флага взаимодействия с ним
FLAG_BEFORE_ERROR	9	Флаг, который вы указали для работы с изображением, необходимо указать после нужного для него ключа
DIVIDE_ERROR	10	Неверные данные для разделения изображения

FLAG_ALL_REACTANGLE_ERROR	11	Неверные данные для обводки прямоугольников или их поиска
BMP_ERROR	12	Неподдерживаемый формат BMP-файла

Соответствующее значение ошибки, записанной в переменной err функции main() выводится в конце работы программы.

В функции ChooseColor() на вход подается указатель на название цвета и на структуру Rgb, в который следует сохранить соответствующие значения компонент. С помощью функции strcmp() проверяется совпадение между названием, введенным пользователем и реализованным списком цветов. Если цвет ни с одним не совпал, возвращается ошибка COLOR_ERROR.

В функции ArgChecks() на вход подается аргумент из массива аргументов командной строки argv[]. Проверяется, что аргумент не является ключом - для этого с помощью функции strcmp() проверяется значение char* arg со всеми возможными (короткими и длинными) названиями опций. Если хоть одно значение совпало, то функция вернет ошибку ARG_ERROR.

Функция CheckArgCircleRadius() принимает на вход координаты центра окружности, радиус, толщину, а также значение высоты и ширины изображения. Функция возвращает ошибку CIRCLE_RADIUS_ERROR в следующих случаях: центр окружности выходит за рамки изображения, радиус 0 или отрицательное число, толщина превосходит по размерам изображение или меньше 1.

Функция CheckArgCircleSquare() принимает на вход координаты левого-верхнего и правого-нижнего угла описанного для окружности квадрата, толщину линии и размеры изображения. По заданным координатам углов проверяется, что фигура - квадрат, также проверяется толщина аналогично

проверке в функции CheckArgCircleRadius(), и в случае несоответствий возвращается ошибка CIRCLE_SQUARE_ERROR.

Функция CheckArgRgbFilter() принимает на вход указатель на строку с компонентом RGB, который необходимо поменять, и значение value. Ошибка RGB_FILTER_ERROR возвращается в том случае, если значение value выходит за рамки ([0;255]) или компонент указан неверно (любая другая буква, кроме r, g, b).

Функция CheckArgDivide() принимает на вход количество частей N и M, на которые следует разделить изображение, толщину линии, и размеры изображения. В случае, если количество частей меньше 1 или превосходит размеры изображения, а также несоответствие толщины линии (аналогично предыдущим функциям), возвращается ошибка DIVIDE_ERROR.

Функция CheckArgFindAllRectangle() принимает на вход толщину линии и размеры изображения. Аналогично предыдущим функциям, в случае несоответствия толщины возвращается ошибка FIND_ALL_RECTANGLE_ERROR.

1.6. Вывод информации об файле. Функции printFileHeader() и printInfoHeader().

Функциям printFileHeader() и printInfoHeader() подаются на вход соответствующие структуры BitmapFileHeader и BitmapInfoHeader и выводятся с помощью функции printf() все поля данных структур.

2. ОБРАБОТКА ИЗОБРАЖЕНИЙ

2.1. RGB-фильтр. Функция rgb_filter().

На вход функции подается указатель на строку, содержащую название компоненты, значение, в которое ее необходимо изменить, размеры изображения по высоте и ширине, массив пикселей.

С помощью оператора switch выбирается соответсвующая компонента, осуществляется проход в цикле по всему изображению и изменяется компонента каждого пикселя на значение value.

2.2. Рисование окружности. Функции pre_DrawCircle() и DrawCircle().

Основной функцией является функция DrawCircle(), на вход которой подается координаты центра вдоль высоты и ширины, радиус окружности, массив пикселей, размеры изображения, толщина линии обводки, флажок fill, указывающий на факт необходимости заливки окружности, и две структуры Rgb с цветом для линии и заливки окружности в случае необходимости.

Функция определяет на основании центра и радиуса координаты левого-верхнего и правого-нижнего угла квадрата, который можно описать около окружности. Далее осуществляется пробег по всем точкам данного квадрата. Игнорируются точки, которые выходят за границы изображения, тем самым позволяя рисовать ту часть окружности, которая находится внутри.

Непосредственно отрисовка окружности осуществляется с помощью формулы окружности: определяются квадраты координат двух точек, определяется радиус окружности, находящейся внутри (область между исходной окружностью и внутренней и есть толщина окружности), проверяется, что точка лежит внутри исходной окружности, но при этом вне внутренней окружности - точка перекрашивается в цвет линии. Для заливки окружности проверяется, что точка внутри внутренней окружности, а также флажок fill имеет значение 1.

Функция pre_DrawCircle вызывается в том случае, если ранее была определена опция -q по рисованию окружности с координатами углов описанного квадрата. Функция преобразует эти координаты в центр окружности, определяет ее радиус и вызывает основную функцию DrawCircle().

2.3. Поиск всех залитых прямоугольников заданного цвета. Функция findAllRectangle().

На вход функции подается двумерный массив position, полученный в результате работы функции createArrPosition() и состоящий следующим образом: на том месте массива (координаты пикселя), на котором пиксель закрашен в цвет, который нужно найти с прямоугольником, стоит 1, в ином случае 0. Размерность массива совпадает с размерностью изображения. Функция также принимает на вход размеры изображения, массив пикселей, цвет линии, которым нужно обвести найденные прямоугольники, и толщина, а также цвет заливки самого прямоугольника.

Определяются переменная S - площадь фигуры, структура Figure border, в которой определы поля для координат 4 вершин прямоугольника, а также координата самой верхней и нижней точки фигуры (с свою очередь эти поля являются структурами Point, в которой и определяется координата по высоте и ширине X и Y соответственно), флажок для обозначения, что искомая исследованная фигура - прямоугольник, и количество найденных прямоугольников, для вывода сообщения в том случае, если прямоугольник не был найден.

Пробегаясь по всему изображению в массиве position, ищется первое вхождение необходимого цвета, обозначаемого 1. В этом случае инициализируются все поля структуры border, вызывается функция fill(), которая ищет все границы вершин фигуры, подсчитывается площадь функцией count_area(), осуществляется проверка на то, что фигура является прямоугольником с помощью функции check_rectangle(). Если фигура является прямоугольником - вызывается функция hightlightRectangle(), которая обводит

его, выводится сообщение об координатах найденного прямоугольника (верхний левый и правый нижний угол). Для следующей итерации площадь зануляется.

Для поиска вершин было реализовано две функции - search_tops() и fill(). Первая функция явилась простой в реализации, однако неэффективной, так как искала вершины рекурсивно, поэтому данная функция была модернизирована функцией fill(). Данная функция инициализирует структуру Point el координатами точек, массив этих точек (размерность массива H*W*2, то есть удвоенный размер картинки, что гарантирует, что он не будет заполнен полностью), переменную count для подсчета количества точек, которые нужно перекрасить, и индекс последнего элемента в массиве. Искомая точка «перекрашивается» в значение 2, в цикле while() осуществляется пробег по всему контуру фигуры до тех пор, пока количество элементов в массиве не обнулится: так, если по 4 сторонам, т.е. соседние пиксели стоят в 1, то координаты этих точек записываются в массив. Внутри такого алгоритма заливки осуществляется поиск границ фигуры - самые левые-верхние, самые нижние и т.д. точки у заданной области. Функция, которой передан указатель на структуру Figure* border, по завершению работы найдет и запишет все нужные вершины и границы.

Функция count_area() пробегается, на основании найденных границ у фигуры, по описанному прямоугольнику и подсчитывает количество точек из массива position, которые уставлены в 2. Таким образом, функция возвращает площадь фигуры.

Функция check_rectangle() принимает на вход границы пройденной фигуры и ее площадь. На основании границ проверяется, что все соответсвующие вершины лежат на одной линии, как и должно быть у прямоугольника, что площадь, подсчитанная ранее, совпадает с площадью, выраженной через длины сторон прямоугольника (разница между границами), а также проверка на то, что площадь не равна 0. По итогу работы этой функции исключаются все ненужные фигуры, которые имеют какие-либо выступы,

дырки, или впринципе не соответсвующие описанию прямоугольника, и возвращает значение 1, если эта фигура удовлетворяет всем условим.

Функция hightlightRectangle() определяет границы описанного прямоугольника, длины которого отстоят от исходного на величину толщины. Осуществляется пробег по координатам описанного прямоугольника, проверяется, что координаты точки лежат в области между исходным и описанным прямоугольником, что точки не лежат вне изображения, и перекрашиваются в необходимый цвет.

2.4. Разделение изображения на N*M частей. Функция Divide().

Для разделения изображения был выбран вариант с проведением линий заданной толщины и цвета. На вход функции подается массив пикселей arr, указатели на структуры BitmapFileHeader и BitmapInfoHeader (bmfh и bmif соответственно), количество частей, на которое нужно разбить изображение, а также толщина и цвет линии.

Так как для разделения изображения необходимо увеличивать изображение и проводить линии между частями, то инициализируются переменные oldHeight (значение старой высоты, полученной из поля height структуры BitmapInfoHeader) и oldWidth. Создаются новые указатели на структуры BitmapFileHeader и BitmapInfoHeader (Newbmfh и Newbmif). Поля, определяющие высоту и ширину в этих структурах, увеличиваются на количество проводимых линий (на единицу меньше, чем число частей), умноженное на толщину линий. Инициализируются переменные newHeight и newWidth. Создается новый массив пикселей newArr, изначальное изображение, т.е. массив arr переводится в новый массив. Затем осуществляется пробег по количеству частей, которых должно быть при разделении. Область, отстоящая от линии с заданной толщиной, перекопируется вверх и вправо для обоих циклов, внутри этих циклов заданы еще одни циклы, которые заполяют место, на котором должна быть линия, цветом этой линии. Таким образом получаем разделенное линиями изображение. Для того, чтобы зафиксировать изменения в

размере файла и самой картинке, инициализируется и выделяется память под структуру newPicture, содержащую поля с массивом пикселей, заголовку и информации об файле. По завершении работы функции изначальные указатели на структуры BitmapFileHeader, BitmapInfoHeader,массив с пикселями Rgb** arr переопределяются полями полученной структуры newPicture, которая сохранена в указатель на эту структуру newFileInfo.

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Вывод справки:

Команда: ./edit -h

Результат:

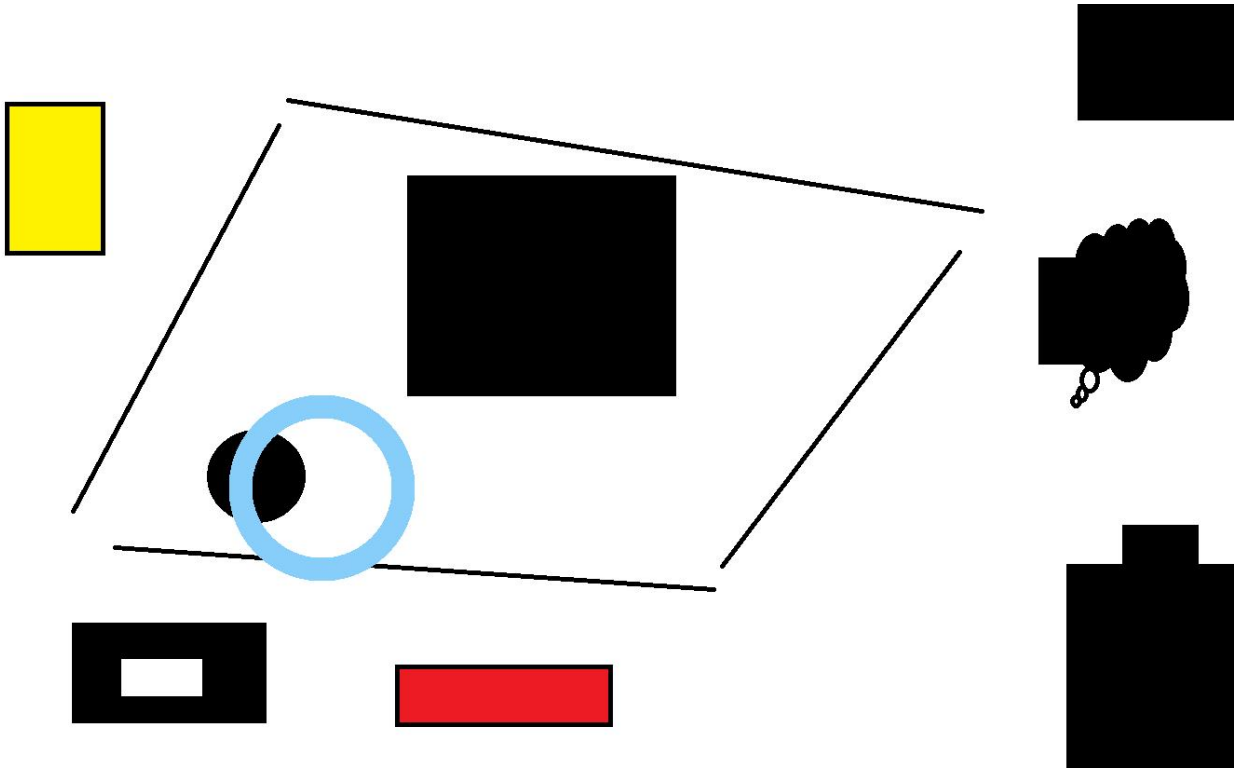
```
alexander@LAPTOP-57T2G9ML:~/my_project$ ./edit -h
Help:
        --open <name>   -o: Считать изображение
        --circle        -c: Нарисовать окружность по следующим условиям:
                --radius <радиус> <высота x> <ширина y> <толщина> <цвет линии>  -r: по радиусу и координатам
                --square <высота x1> <ширина y1> (координаты верхнего левого угла) <высота x2> <ширина y2> (координаты нижнего правого угла) <толщина> <цвет линии>     -q:
по описанному квадрату
                        Дополнительные флаги для работы с окружностью:
                        --fill <color>  -f: Выбор цвета заливки окружности
        --filter <компонент Rgb> <значение компонента>  -p: Rgb-фильтр
        --divide <количество частей вдоль высоты N> <количество частей вдоль ширины M> <толщина> <цвет линии>   -d: Разделяет изображение линиями на N*M частей
        --findAllRectangle <цвет заливки> <цвет обводки> <толщина обводки>      -a: поиск всех прямоугольников заданного цвета и их обводка
        --information   -i: Вывести информацию об считанном файле
        --help  -h: Открыть подсказку
                Доступны следующие цвета:
                black, pirple, blue, light_blue, green, yellow, orange, red, gray, white
        Пример работы программы:
                ./edit -o your.bmp -c -r 100 300 400 25 pirple
                ./edit -o your.bmp -c -r 100 300 400 25 pirple -f green
                ./edit -o your.bmp --circle --square 400 200 300 300 100 black -f pirple
                ./edit -o your.bmp --circle --square 400 200 300 300 25 black --fill pirple
                ./edit -o your.bmp -p r 255
                ./edit -o your.bmp --divide 2 2 25 yellow
                ./edit --open your.bmp --findAllRectangle black red 25
                ./edit -o your.bmp -i
                ./edit --help
        результат сохраняется в файл out.bmp
```

Рисование окружности по радиусу:

Команда: ./edit --open ex1.bmp -c --radius 100 300 400 25 light_blue

Исходный файл: ex1.bmp

Результат:

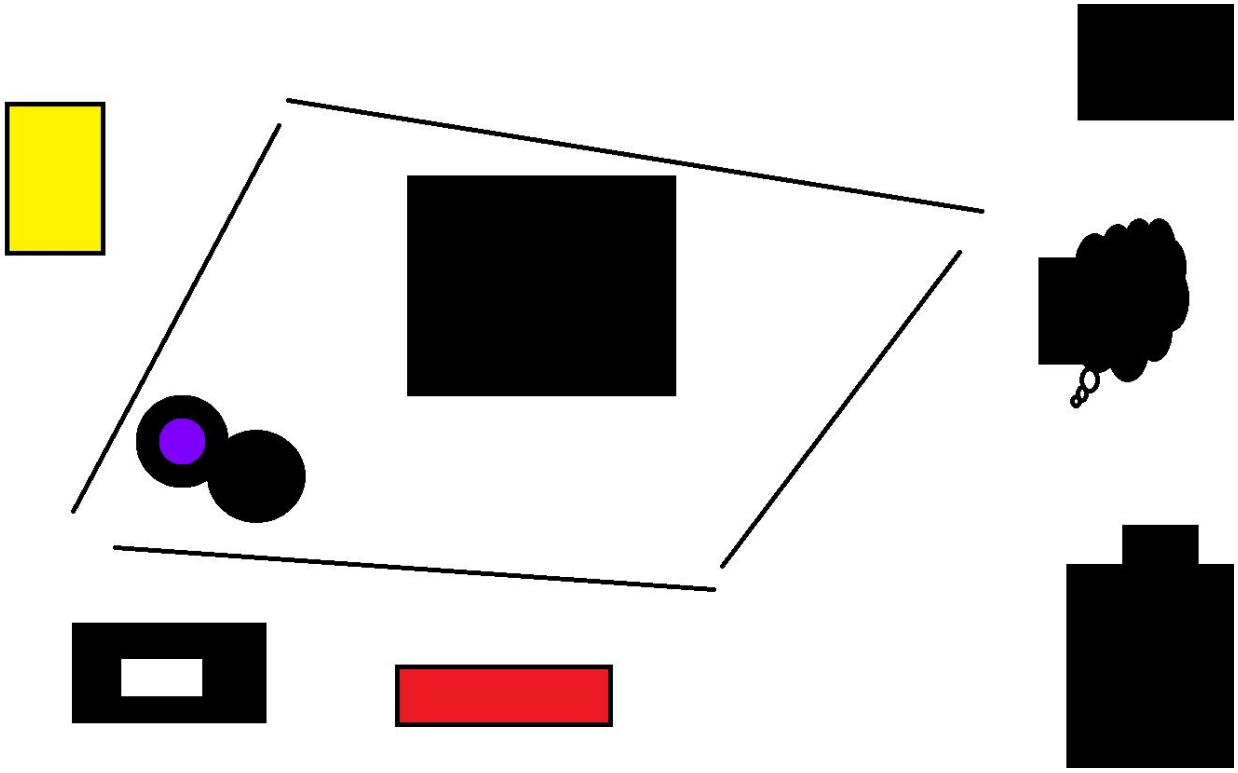


Рисование окружности по описанному квадрату:

Команда: ./edit --open ex1.bmp -c -q 400 200 300 300 25 black -f pirple

Исходный файл: ex1.bmp

Результат:



Поиск всех прямоугольников и их обводка:

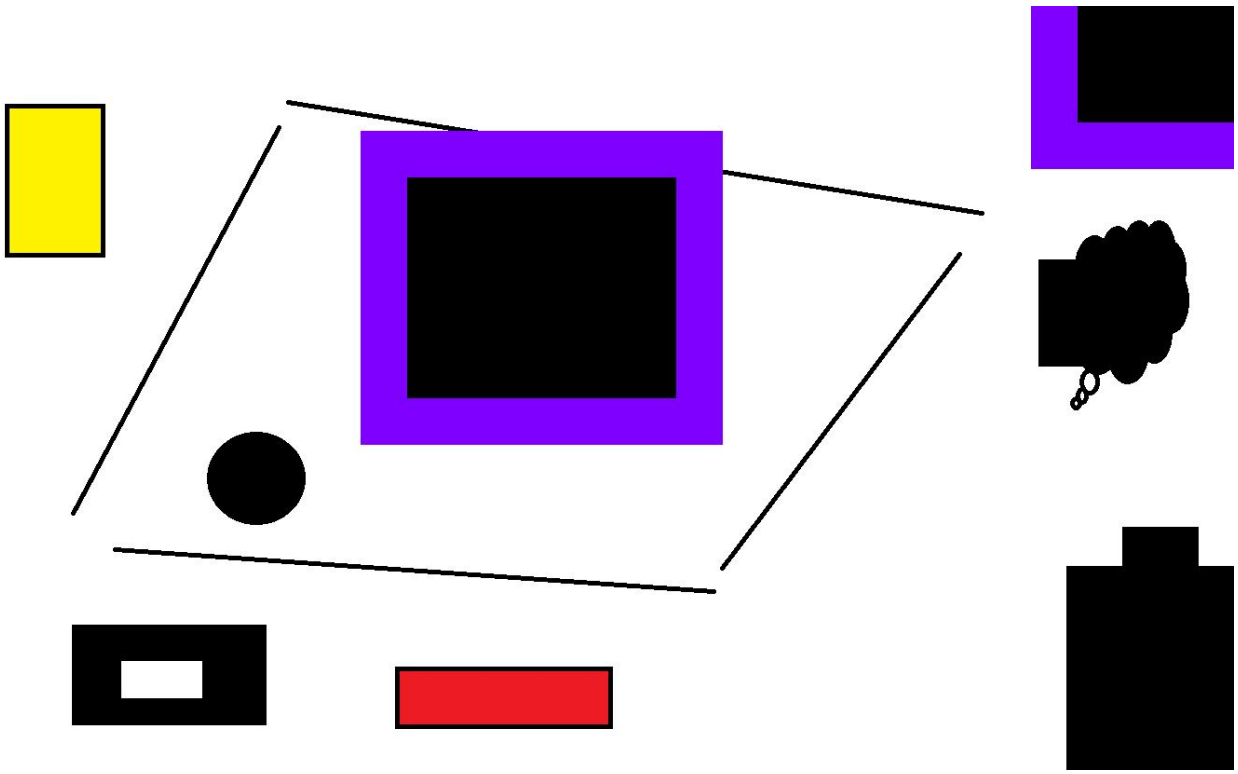
Команда: ./edit --open ex1.bmp -a black pirple 50

Исходный файл: ex1.bmp

Результат:

```
alexander@LAPTOP-57T2G9ML:~/my_project$ ./edit --open ex1.bmp -a black pirple 50

Координаты левого-верхнего и правого-нижнего угла найденных прямоугольников:
Левый-верхний:492 635, Правый-нижний:780 399
Левый-верхний:1212 819, Правый-нижний:1379 695
```

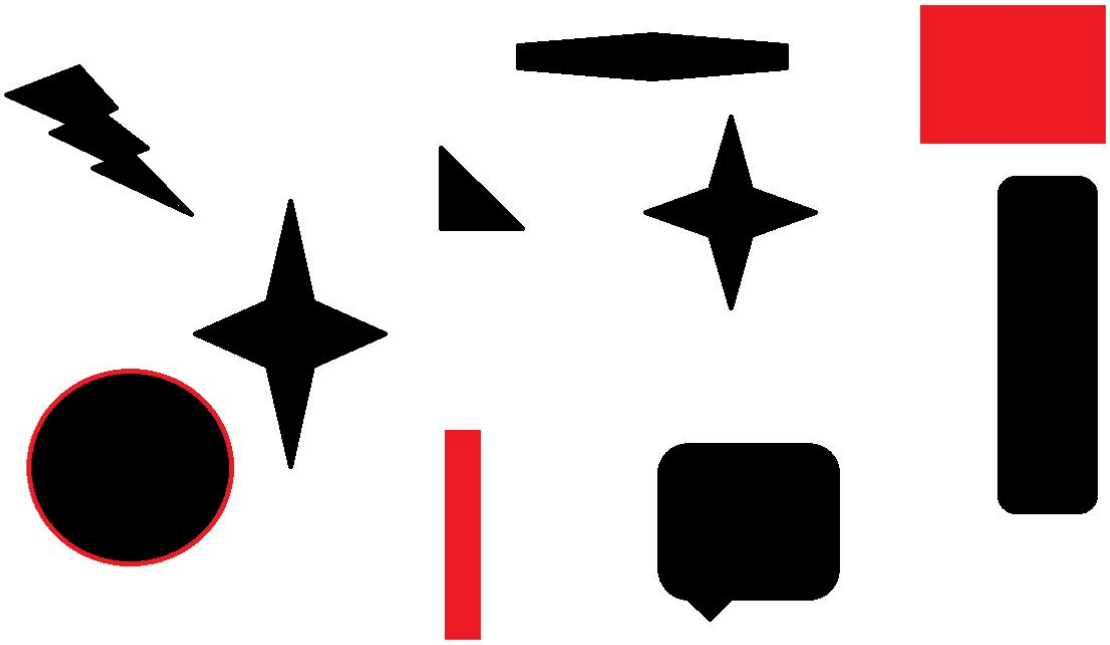


Отсутствие прямоугольников для обводки:

Команда: ./edit --open ex2.bmp -a black pirple 50

Исходный файл: ex2.bmp

Результат:

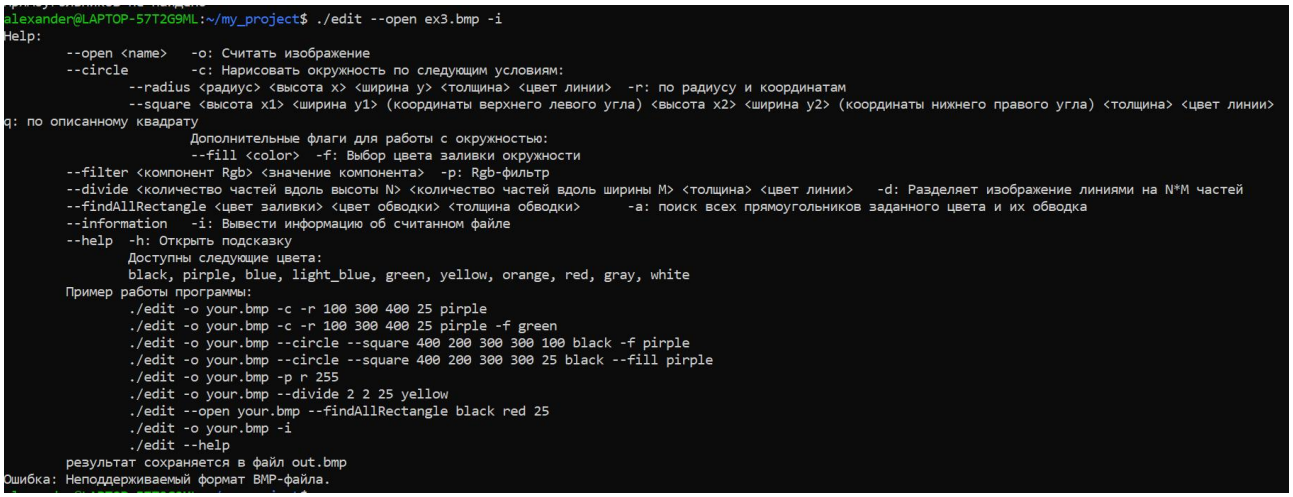


Неверный формат файла-BMP:

Команда: ./edit --open ex3. -i

Исходный файл: ex3.bmp

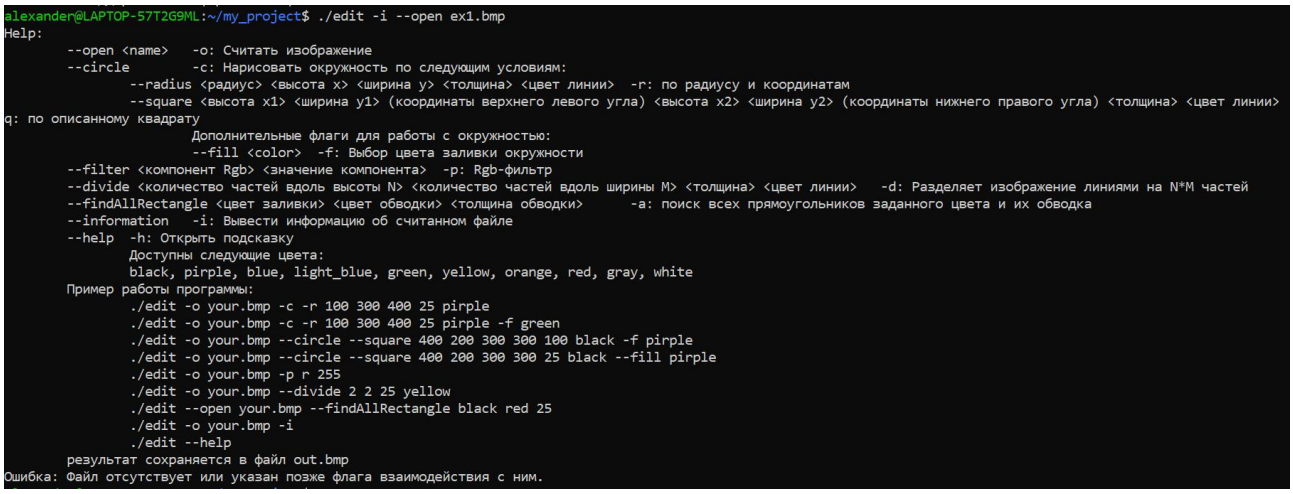
Результат:



Вывод справки до ключа --open:

Команда: ./edit -i --open ex1.bmp

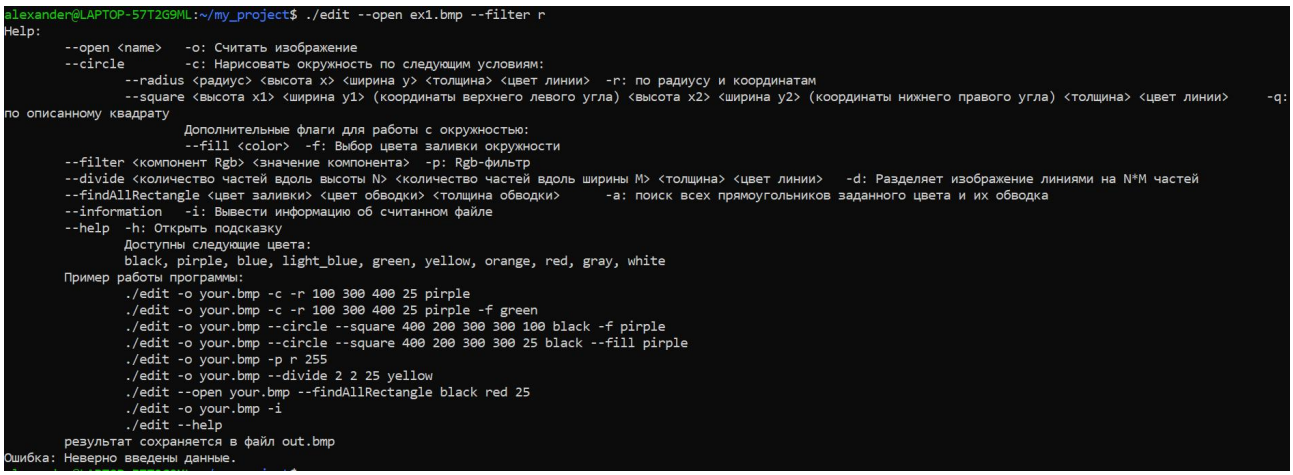
Результат:



Неверное число аргументов:

Команда: ./edit --open ex1.bmp --filter r

Результат:

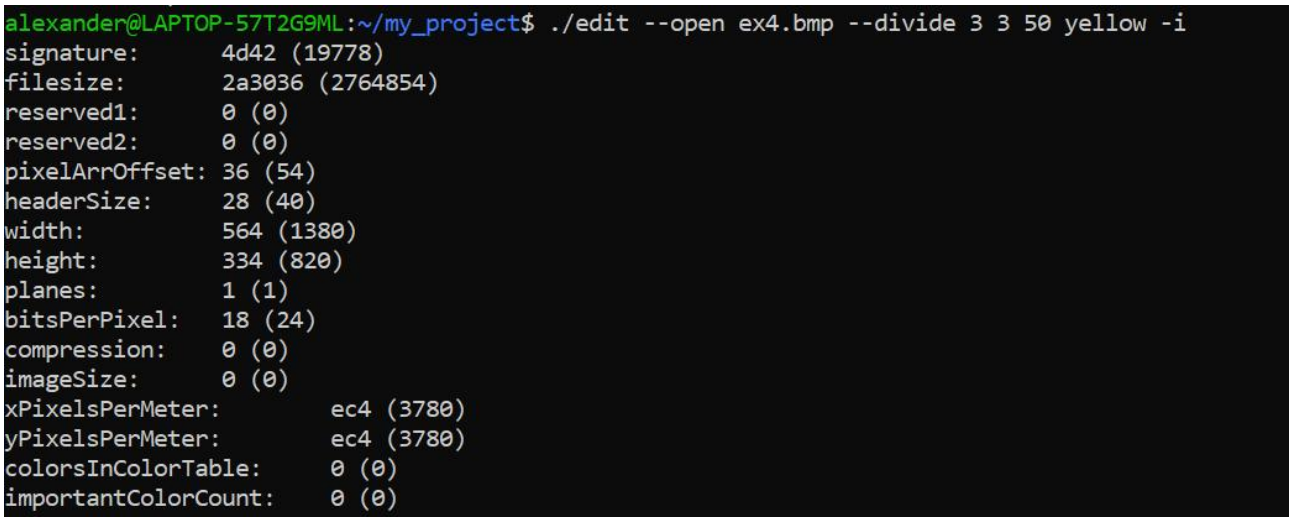


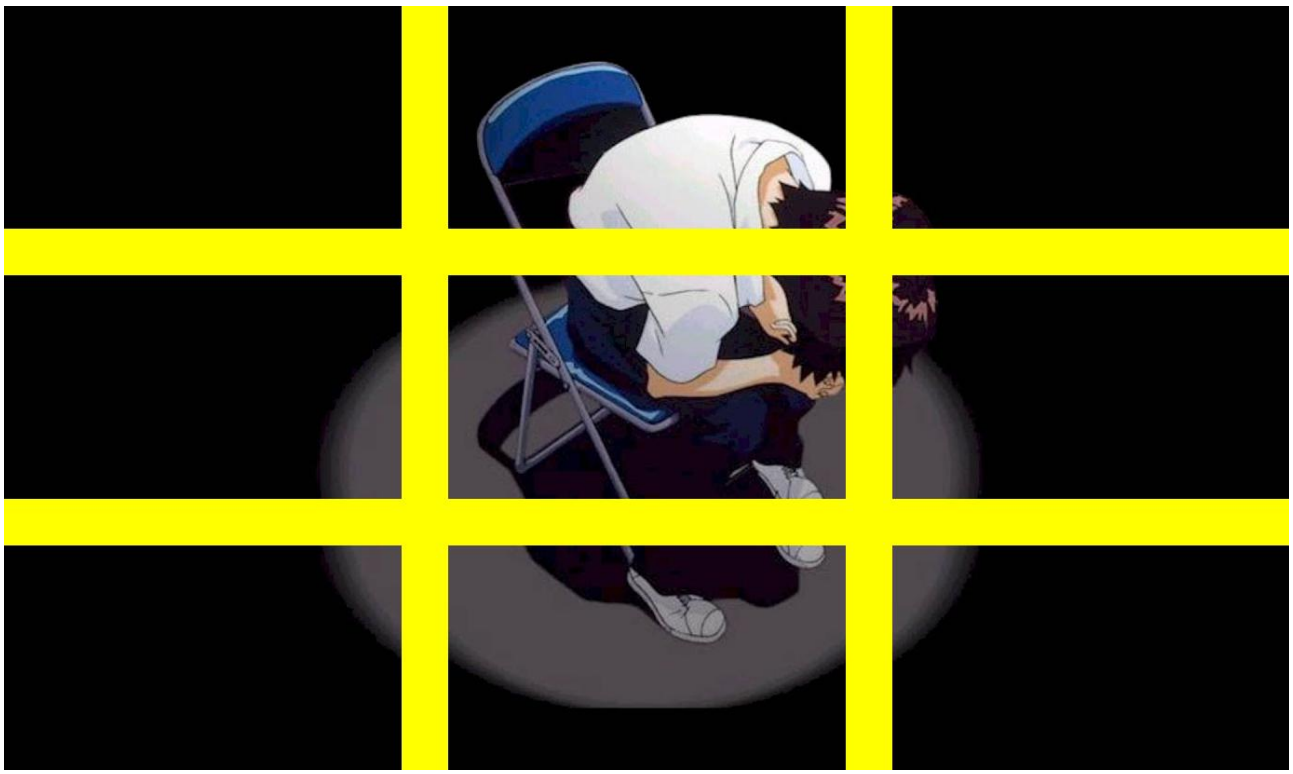
Разделение изображения:

Команда: ./edit --open ex4.bmp --divide 3 3 50 yellow -I

Исходный файл: ex4.bmp

Результат:





Неверный аргумент для rgb-фильтра:

Команда: ./edit --open ex1.bmp -p r 256

Результат:

```
alexander@LAPTOP-57T2G9ML:~/my_project$ ./edit --open ex1.bmp -p r 256
Help:
        --open <name>   -o: Считать изображение
        --circle        -c: Нарисовать окружность по следующим условиям:
                --radius <радиус> <высота x> <ширина y> <толщина> <цвет линии>  -r: по радиусу и координатам
                --square <высота x1> <ширина y1> (координаты верхнего левого угла) <высота x2> <ширина y2> (координаты нижнего правого угла) <толщина> <цвет линии>
q: по описанному квадрату
                        Дополнительные флаги для работы с окружностью:
                        --fill <color>  -f: Выбор цвета заливки окружности
        --filter <компонент Rgb> <значение компонента>  -p: Rgb-фильтр
        --divide <количество частей вдоль высоты N> <количество частей вдоль ширины M> <толщина> <цвет линии>   -d: Разделяет изображение линиями на N*M частей
        --findAllRectangle <цвет заливки> <цвет обводки> <толщина обводки>      -a: поиск всех прямоугольников заданного цвета и их обводка
        --information   -i: Вывести информацию об считанном файле
        --help  -h: Открыть подсказку
                Доступны следующие цвета:
                black, pirple, blue, light_blue, green, yellow, orange, red, gray, white
        Пример работы программы:
                ./edit -o your.bmp -c -r 100 300 400 25 pirple
                ./edit -o your.bmp -c -r 100 300 400 25 pirple -f green
                ./edit -o your.bmp --circle --square 400 200 300 300 100 black -f pirple
                ./edit -o your.bmp --circle --square 400 200 300 300 25 black --fill pirple
                ./edit -o your.bmp -p r 255
                ./edit -o your.bmp --divide 2 2 25 yellow
                ./edit --open your.bmp --findAllRectangle black red 25
                ./edit -o your.bmp -i
                ./edit --help
        результат сохраняется в файл out.bmp
Ошибка: Неверные данные для Rgb-фильтра.
```

Совершение действия до указания файла:

Команда: ./edit -i -c -r 100 200 300 25 pirple --open ex1.bmp

Результат:

```
alexander@LAPTOP-57T2G9ML:~/my_project$ ./edit -i -c -r 100 200 300 25 pirple --open ex1.bmp
Help:
        --open <name>   -o: Считать изображение
        --circle        -c: Нарисовать окружность по следующим условиям:
                --radius <радиус> <высота x> <ширина y> <толщина> <цвет линии>  -r: по радиусу и координатам
                --square <высота x1> <ширина y1> (координаты верхнего левого угла) <высота x2> <ширина y2> (координаты нижнего правого угла) <толщина> <цвет линии>
q: по описанному квадрату
                        Дополнительные флаги для работы с окружностью:
                        --fill <color>  -f: Выбор цвета заливки окружности
        --filter <компонент Rgb> <значение компонента>  -p: Rgb-фильтр
        --divide <количество частей вдоль высоты N> <количество частей вдоль ширины M> <толщина> <цвет линии>   -d: Разделяет изображение линиями на N*M частей
        --findAllRectangle <цвет заливки> <цвет обводки> <толщина обводки>      -a: поиск всех прямоугольников заданного цвета и их обводка
        --information   -i: Вывести информацию об считанном файле
        --help  -h: Открыть подсказку
                Доступны следующие цвета:
                black, pirple, blue, light_blue, green, yellow, orange, red, gray, white
        Пример работы программы:
                ./edit -o your.bmp -c -r 100 300 400 25 pirple
                ./edit -o your.bmp -c -r 100 300 400 25 pirple -f green
                ./edit -o your.bmp --circle --square 400 200 300 300 100 black -f pirple
                ./edit -o your.bmp --circle --square 400 200 300 300 25 black --fill pirple
                ./edit -o your.bmp -p r 255
                ./edit -o your.bmp --divide 2 2 25 yellow
                ./edit --open your.bmp --findAllRectangle black red 25
                ./edit -o your.bmp -i
                ./edit --help
        результат сохраняется в файл out.bmp
Ошибка: Файл отсутствует или указан позже флага взаимодействия с ним.
```

RGB-фильтр:

Команда: ./edit --open ex5.bmp -p r 0

Исходный файл: ex5.bmp

Результат:

Поиск большого прямоугольника (проверка на отсутствие ошибок в реализации)

Команда: ./edit --open ex5.bmp -a red light_blue 100

Исходный файл: ex5.bmp

Результат:

ЗАКЛЮЧЕНИЕ

По итогам курсовой работы была создана программа, имеющая CLI и реализующая функции по обработке BMP-изображений. Для выполнения поставленных задач была изучена структура хранения сведений об изображении, BitmapHeader и BitmapInfo, алгоритм заливки, особенности хранения массива пикселей (RGB) в изображениях, считывания файлов и их вывод. По итогу программа способна реализовать следующий функционал: нарисовать окружность ко заданному радиусу и центру, или же по координатам описанного квадрата, при этом окружность может быть залитой или нет, фильтр-RGB, разделение изображения на N*M частей, поиск всех залитых прямоугольников заданного цвета и их обводка, вывод информации об файлах.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Инструкция по запуску: запустить Makefile, запустить исполняемый файл ./edit

Название файла: Makefike

```
ALL:
	GCC MAIN.C -O EDIT
```

Название файла: main.c

```
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>
#INCLUDE <STRING.H>
#INCLUDE <MATH.H>
#INCLUDE <UNISTD.H>
#INCLUDE <GETOPT.H>

#DEFINE NO_ERROR 0
#DEFINE ARG_ERROR 1
#DEFINE OPEN_IMAGE_ERROR 2
#DEFINE KEY_ERROR 3
#DEFINE COLOR_ERROR 4
#DEFINE CIRCLE_RADIUS_ERROR 5
#DEFINE CIRCLE_SQUARE_ERROR 6
#DEFINE RGB_FILTER_ERROR 7
#DEFINE INFORMATION_ERROR 8
#DEFINE FLAG_BEFORE_ERROR 9
#DEFINE DIVIDE_ERROR 10
#DEFINE FIND_ALL_RECTANGLE_ERROR 11
#DEFINE BMP_ERROR 12

CHAR* ERROR_MSG[] =
{
    "ОШИБОК НЕТ",
    "НЕВЕРНО ВВЕДЕНЫ ДАННЫЕ",
    "ИЗОБРАЖЕНИЕ НЕ НАЙДЕНО",
    "НЕВЕРНО ВВЕДЕН КЛЮЧ ИЛИ ЕГО АРГУМЕНТЫ ОТСУТСТВУЮТ",
    "НЕИЗВЕСТНЫЙ ЦВЕТ",
    "ПО ВАШИМ ДАННЫМ НЕВОЗМОЖНО ПОСТРОИТЬ КРУГ. ПЕРЕПРОВЕРЬТЕ ВХОДНЫЕ
ПАРАМЕТРЫ",
    "НЕВЕРНО ВВЕДЕНЫ КООРДИНАТЫ КВАДРАТА",
    "НЕВЕРНЫЕ ДАННЫЕ ДЛЯ RGB-ФИЛЬТРА",
    "ФАЙЛ ОТСУТСТВУЕТ ИЛИ УКАЗАН ПОЗЖЕ ФЛАГА ВЗАИМОДЕЙСТВИЯ С НИМ",
    "ФЛАГ, КОТОРЫЙ ВЫ УКАЗАЛИ ДЛЯ РАБОТЫ С ИЗОБРАЖЕНИЕМ, НЕОБХОДИМО
УКАЗАТЬ ПОСЛЕ НУЖНОГО ДЛЯ НЕГО КЛЮЧА",
    "НЕВЕРНЫЕ ДАННЫЕ ДЛЯ РАЗДЕЛЕНИЯ ИЗОБРАЖЕНИЯ",
    "НЕВЕРНЫЕ ДАННЫЕ ДЛЯ ОБВОДКИ ПРЯМОУГОЛЬНИКОВ ИЛИ ИХ ПОИСКА",
    "НЕПОДДЕРЖИВАЕМЫЙ ФОРМАТ BMP-ФАЙЛА"
};


#PRAGMA PACK (PUSH, 1) //
TYPEDEF STRUCT BITMAPFILEHEADER
{
```

```
    UNSIGNED SHORT SIGNATURE; // ПОЗВОЛЯЕТ ОПРЕДЕЛИТЬ ТИП ФАЙЛА
    UNSIGNED INT FILESIZE; // РАЗМЕР ФАЙЛА
    UNSIGNED SHORT RESERVED1; // ДОЛЖЕН БЫТЬ 0
    UNSIGNED SHORT RESERVED2; // ДОЛЖЕН БЫТЬ 0
    UNSIGNED INT PIXELARROFFSET; // ПОКАЗЫВАЕТ, ГДЕ НАЧИНАЕТСЯ САМ
БИТОВЫЙ МАССИВ ОТНОСИТЕЛЬНО НАЧАЛА ФАЙЛА, КОТОРЫЙ И ОПИСЫВАЕТ КАРТИНКУ
} BITMAPFILEHEADER;

TYPEDEF STRUCT BITMAPINFOHEADER
{
    UNSIGNED INT HEADERSIZE; // РАЗМЕР СТРУКТУРЫ
    UNSIGNED INT WIDTH; // ШИРИНА КАРТИНКИ В ПИКСЕЛЯХ
    UNSIGNED INT HEIGHT; // ВЫСОТА КАРТИНКИ В ПИКСЕЛЯХ
    UNSIGNED SHORT PLANES; // ЗАДАЕТ КОЛИЧЕСТВО ПЛОСКОСТЕЙ, ПОКА
ВСЕГДА УСТАНАВЛИВАЕТСЯ В 1
    UNSIGNED SHORT BITSPERPIXEL; // КОЛИЧЕСТВО БИТ НА ОДИН ПИКСЕЛЬ
    UNSIGNED INT COMPRESSION; // ТИП СЖАТИЯ, ОБЫЧНО НЕСЖАТЫЕ КАРТИНКИ
    UNSIGNED INT IMAGESIZE; // РАЗМЕР КАРТИНКИ В БАЙТАХ; ЕСЛИ
ИЗОБРАЖЕНИЕ НЕСЖАТО, ТО ЗДЕСЬ ДОЛЖЕН БЫТЬ ЗАПИСАН 0
    UNSIGNED INT XPIXELSPERMETER; // ГОРИЗОНАТАЛЬНОЕ РАЗРЕШЕНИЕ
(ПИКСЕЛЬ НА МЕТР)
    UNSIGNED INT YPIXELSPERMETER; // ВЕРТИКАЛЬНОЕ РАЗРЕШЕНИЕ (ПИКСЕЛЬ
НА МЕТР)
    UNSIGNED INT COLORSINCOLORTABLE; // КОЛИЧЕСТВО ИСПОЛЬЗУЕМЫХ ЦВЕТОВ
ИЗ ТАБЛИЦЫ, ЕСЛИ ЗАПИСАН 0, ТО В РАСТРЕ ИСПОЛЬЗУЕТСЯ МАКСИМАЛЬНО
ВОЗМОЖНОЕ КОЛИЧЕСТВО ЦВЕТОВ
    UNSIGNED INT IMPORTANTCOLORCOUNT; // КОЛИЧЕСТВО ВАЖНЫХ ЦВЕТОВ,
ЕСЛИ ЭТО ЗНАЧЕНИЕ 0, ТО ВСЕ ЦВЕТА СЧИТАЮТСЯ ВАЖНЫМИ
} BITMAPINFOHEADER;

TYPEDEF STRUCT RGB
{
    UNSIGNED CHAR B;
    UNSIGNED CHAR G;
    UNSIGNED CHAR R;
} RGB;
#PRAGMA PACK(POP) //

VOID PRINTFILEHEADER (BITMAPFILEHEADER HEADER){ // ВЫВОДИТ
СООТВЕТСВУЮЩУЮ ИНФОРМАЦИЮ СТРУКТУРЫ BITMAPFILEHEADER
    PRINTF("SIGNATURE:\T%X (%HU)\N", HEADER.SIGNATURE,
HEADER.SIGNATURE);
    PRINTF("FILESIZE:\T%X (%U)\N", HEADER.FILESIZE, HEADER.FILESIZE);
     PRINTF("RESERVED1:\T%X (%HU)\N", HEADER.RESERVED1,
HEADER.RESERVED1);
     PRINTF("RESERVED2:\T%X (%HU)\N", HEADER.RESERVED2,
HEADER.RESERVED2);
     PRINTF("PIXELARROFFSET:\T%X (%U)\N", HEADER.PIXELARROFFSET,
HEADER.PIXELARROFFSET);
}

VOID PRINTINFOHEADER(BITMAPINFOHEADER HEADER){ // ВЫВОДИТ
СООТВЕТСТВУЮЩУЮ ИНФФОРМАЦИЮ ПО ФАЙЛУ BMP (СТРУКТУРА BITMAPINFOHEADER)
     PRINTF("HEADERSIZE:\T%X (%U)\N", HEADER.HEADERSIZE,
HEADER.HEADERSIZE);
     PRINTF("WIDTH:     \T%X (%U)\N", HEADER.WIDTH, HEADER.WIDTH);
     PRINTF("HEIGHT:    \T%X (%U)\N", HEADER.HEIGHT, HEADER.HEIGHT);
```

```
     PRINTF("PLANES:    \T%X (%HU)\N", HEADER.PLANES, HEADER.PLANES);
     PRINTF("BITSPERPIXEL:\T%X (%HU)\N", HEADER.BITSPERPIXEL,
HEADER.BITSPERPIXEL);
     PRINTF("COMPRESSION:\T%X (%U)\N", HEADER.COMPRESSION,
HEADER.COMPRESSION);
     PRINTF("IMAGESIZE:\T%X (%U)\N", HEADER.IMAGESIZE,
HEADER.IMAGESIZE);
     PRINTF("XPIXELSPERMETER:\T%X (%U)\N", HEADER.XPIXELSPERMETER,
HEADER.XPIXELSPERMETER);
     PRINTF("YPIXELSPERMETER:\T%X (%U)\N", HEADER.YPIXELSPERMETER,
HEADER.YPIXELSPERMETER);
     PRINTF("COLORSINCOLORTABLE:\T%X (%U)\N",
HEADER.COLORSINCOLORTABLE, HEADER.COLORSINCOLORTABLE);
     PRINTF("IMPORTANTCOLORCOUNT:\T%X (%U)\N",
HEADER.IMPORTANTCOLORCOUNT, HEADER.IMPORTANTCOLORCOUNT);
}


INT CHECKRGBSTRUCT(RGB COLOR_1, RGB COLOR_2){
    IF (COLOR_1.R == COLOR_2.R && COLOR_1.G == COLOR_2.G && COLOR_1.B
== COLOR_2.B){
        RETURN 1;
    }
    ELSE
        RETURN 0;
}

VOID CREATEARRPOSITION(RGB** ARR, RGB FIND_COLOR, INT H, INT W, INT**
POSITION){
    FOR (INT I = 0; I<H; I++){
        FOR (INT J = 0; J<W; J++){
            IF (CHECKRGBSTRUCT(FIND_COLOR, ARR[I][J])){
                POSITION[I][J] = 1;
            }
            ELSE
                POSITION[I][J] = 0;
        }
    }
}

TYPEDEF STRUCT POINT{
     INT X; // КООРДИНАТА ВДОЛЬ ШИРИНЫ
     INT Y; // КООРДИНАТА ВДОЛЬ ВЫСОТЫ
} POINT;

TYPEDEF STRUCT FIGURE{
     POINT TOP1; // ВЕРХНЯЯ ЛЕВАЯ ВЕРШИНА
     POINT TOP2; // ВЕРХНЯЯ ПРАВАЯ ВЕРШИНА
     POINT TOP3; // НИЖНЯЯ ПРАВАЯ ВЕРШИНА
     POINT TOP4; // НИЖНЯЯ ЛЕВАЯ ВЕРШИНА
    POINT MAXTOP; //КООРДИНАТЫ НАИБОЛЬШЕГО X И Y
    POINT MINTOP; // КООРДИНАТЫ НАИМЕНЬШЕГО X И Y
} FIGURE;

VOID FILL(INT Y, INT X, FIGURE* BORDER, INT** POSITION, INT H, INT W)
{
    POINT EL = {X, Y};
```

```
    POINT* VALUE = MALLOC(SIZEOF(POINT)*H*W*2);
    INT COUNT = 0;
    INT TOP_INDEX = -1;
    VALUE[TOP_INDEX+1] = EL; TOP_INDEX++; COUNT++; //PUSH
    POSITION[EL.Y][EL.X] = 2;
    WHILE(COUNT != 0)
    {
        EL = VALUE[TOP_INDEX]; TOP_INDEX--; COUNT--; //POP
        POSITION[EL.Y][EL.X] = 2; //ЗАКРАСИТЬ ПИКСЕЛ НОВЫМ ЦВЕТОМ

    IF((EL.Y >= BORDER->TOP1.Y) && (EL.X <= BORDER->TOP1.X)) { //ПОИСК
ЛВ
        BORDER->TOP1.X = EL.X;
        BORDER->TOP1.Y = EL.Y;
    }

    IF((EL.Y >= BORDER->TOP2.Y) && (EL.X >= BORDER->TOP2.X)) { //ПОИСК
ПВ
        BORDER->TOP2.X = EL.X;
        BORDER->TOP2.Y = EL.Y;
    }

    IF((EL.Y <= BORDER->TOP3.Y) && (EL.X >= BORDER->TOP3.X)) { //ПОИСК
ПН
        BORDER->TOP3.X = EL.X;
        BORDER->TOP3.Y = EL.Y;
    }

    IF((EL.Y <= BORDER->TOP4.Y) && (EL.X <= BORDER->TOP4.X)) { //ПОИСК
ЛН
        BORDER->TOP4.X = EL.X;
        BORDER->TOP4.Y = EL.Y;
    }

    IF ((EL.Y <= BORDER->MINTOP.Y)){ //ПОИСК САМОЙ НИЗКОЙ ВЫСОТЫ
        BORDER->MINTOP.Y = EL.Y;
    }

    IF ((EL.X <= BORDER->MINTOP.X)){ //ПОИСК САМОЙ ЛЕВОЙ ГРАНИЦЫ
        BORDER->MINTOP.X = EL.X;
    }

    IF ((EL.Y >= BORDER->MAXTOP.Y)){ //ПОИСК САМОЙ ВЫСОКОЙ ВЫСОТЫ
        BORDER->MAXTOP.Y = EL.Y;
    }

    IF ((EL.X >= BORDER->MAXTOP.X)){ //ПОИСК САМОЙ ПРАВОЙ ГРАНИЦЫ
        BORDER->MAXTOP.X = EL.X;
    }

        IF(EL.X>0 && POSITION[EL.Y][EL.X-1]==1){
            EL.X = EL.X-1;
            EL.Y = EL.Y;
            VALUE[TOP_INDEX+1] = EL; TOP_INDEX++; COUNT++; //PUSH
        }
        IF(EL.X<(W-1) && POSITION[EL.Y][EL.X+1]==1){
```

```
            EL.X = EL.X+1;
            EL.Y = EL.Y;
            VALUE[TOP_INDEX+1] = EL; TOP_INDEX++; COUNT++; //PUSH
        }
        IF(EL.Y>0 && POSITION[EL.Y-1][EL.X]==1){
            EL.X = EL.X;
            EL.Y = EL.Y-1;
            VALUE[TOP_INDEX+1] = EL; TOP_INDEX++; COUNT++; //PUSH
        }

        IF(EL.Y<(H-1) && POSITION[EL.Y+1][EL.X]==1){
            EL.X = EL.X;
            EL.Y = EL.Y+1;
            VALUE[TOP_INDEX+1] = EL; TOP_INDEX++; COUNT++; //PUSH
        }
    }
    FREE(VALUE);
    RETURN;
}

VOID SEARCH_TOPS(INT Y, INT X, FIGURE* BORDER, INT** POSITION, INT H,
INT W){
    POSITION[Y][X] = 2; //ПРОЙДЕННУЮ ТОЧКУ 1 ПОМЕЧАЕМ 2
    IF((Y >= BORDER->TOP1.Y) && (X <= BORDER->TOP1.X)) { //ПОИСК ЛВ
        BORDER->TOP1.X = X;
        BORDER->TOP1.Y = Y;
    }
    IF((Y >= BORDER->TOP2.Y) && (X >= BORDER->TOP2.X)) { //ПОИСК ПВ
        BORDER->TOP2.X = X;
        BORDER->TOP2.Y = Y;
    }
    IF((Y <= BORDER->TOP3.Y) && (X >= BORDER->TOP3.X)) { //ПОИСК ПН
        BORDER->TOP3.X = X;
        BORDER->TOP3.Y = Y;
    }
    IF((Y <= BORDER->TOP4.Y) && (X <= BORDER->TOP4.X)) { //ПОИСК ЛН
        BORDER->TOP4.X = X;
        BORDER->TOP4.Y = Y;
    }

    IF ((Y <= BORDER->MINTOP.Y)){ //ПОИСК САМОЙ НИЗКОЙ ВЫСОТЫ
        BORDER->MINTOP.Y = Y;
    }
    IF ((X <= BORDER->MINTOP.X)){ //ПОИСК САМОЙ ЛЕВОЙ ГРАНИЦЫ
        BORDER->MINTOP.X = X;
    }
    IF ((Y >= BORDER->MAXTOP.Y)){ //ПОИСК САМОЙ ВЫСОКОЙ ВЫСОТЫ
        BORDER->MAXTOP.Y = Y;
    }
    IF ((X >= BORDER->MAXTOP.X)){ //ПОИСК САМОЙ ПРАВОЙ ГРАНИЦЫ
        BORDER->MAXTOP.X = X;
    }

    //РЕКУРСИВНО ПРОХОДИМСЯ ПО СОСЕДНИМ ТОЧКАМ, ПРОВЕРЯЯ, ЧТО ОНИ
ВХОДЯТ В ГРАНИЦЫ ИЗОБРАЖЕНИЯ
    IF((X < (W-1)) && (POSITION[Y][X+1] == 1)){
        SEARCH_TOPS(Y,X+1,BORDER,POSITION, H, W);
```

```
    }
    IF((Y < (H-1)) && (POSITION[Y+1][X] == 1)){
        SEARCH_TOPS(Y+1,X,BORDER,POSITION, H, W);
    }
    IF((X > 0) && (POSITION[Y][X-1] == 1)){
        SEARCH_TOPS(Y,X-1,BORDER,POSITION, H, W);
    }
    IF((Y > 0) && (POSITION[Y-1][X] == 1)){
        SEARCH_TOPS(Y-1,X,BORDER,POSITION, H, W);
    }

    RETURN;
}

LONG LONG INT COUNT_AREA(FIGURE BORDER, INT** POSITION){
    LONG LONG INT S = 0; //ИНИЦИАЛИЗАЦИЯ ПЛОЩАДИ
    FOR (INT I = BORDER.MINTOP.Y; I <  BORDER.MAXTOP.Y;
I++){ //ПРОХОДИМ ПО ОПИСАННОМУ ПРЯМОУГОЛЬНИКУ (ВЕРШИНЫ ОПРЕДЕЛЕНЫ МАКС
И МИН ГРАНИЦАМИ ФИГУРЫ)
        FOR (INT J = BORDER.MINTOP.X; J < BORDER.MAXTOP.X; J++){
            IF (POSITION[I][J] == 2){ //ЕСЛИ ТОЧКА БЫЛА ПРОЙДЕНА В
SEARCH_TOPS
                S++; // ТО УЧИТЫВАЕМ ЕЕ КАК ТОЧКУ ВНУТРИ ФИГУРЫ
            }
        }
    }
    RETURN S;
}

INT CHECK_RECTANGLE(FIGURE BORDER, INT S){
    IF (BORDER.TOP1.X != BORDER.MINTOP.X || BORDER.TOP1.Y !=
BORDER.MAXTOP.Y){ //ВЕРШИНА ЛВ НЕ СООТВЕТСТВУЕТ КРАЙНИМ ГРАНИЦАМ
ФИГУРЫ
        RETURN 0;
    }
    ELSE IF (BORDER.TOP2.X != BORDER.MAXTOP.X || BORDER.TOP2.Y !=
BORDER.MAXTOP.Y){ // -//-
        RETURN 0;
    }
    ELSE IF (BORDER.TOP3.X != BORDER.MAXTOP.X || BORDER.TOP3.Y !=
BORDER.MINTOP.Y){ // -//-
        RETURN 0;
    }
    ELSE IF (BORDER.TOP4.X != BORDER.MINTOP.X || BORDER.TOP4.Y !=
BORDER.MINTOP.Y){ // -//-
        RETURN 0;
    }
    ELSE IF (S != ((BORDER.MAXTOP.X - BORDER.MINTOP.X) *
(BORDER.MAXTOP.Y - BORDER.MINTOP.Y))){ //ПЛОЩАДЬ, ПОЛУЧЕННАЯ ИСХОДЯ ИЗ
ВЕРШИН ПРЯМОУГОЛЬНИКА СВЕРЯЕТСЯ С ПЛОЩАДЬЮ ПРОЙДЕННОЙ ФИГУРЫ
        RETURN 0;
    }
    ELSE IF (S == 0){
        RETURN 0;
    }
    ELSE
        RETURN 1; //ВСЕ ОК. ПРЯМОУГОЛЬНИК
```

```
}

VOID HIGHTLIGHTRECTANGLE(RGB** ARR, RGB COLOR_LINE, INT THICKNESS,
FIGURE BORDER, INT H, INT W){
    INT XL = BORDER.MINTOP.X - THICKNESS; // ЛЕВАЯ ГРАНИЦА ОПИСАННОГО
ПРЯМОУГОЛЬНИКА
    INT XR = BORDER.MAXTOP.X + THICKNESS; // ПРАВАЯ -//-
    INT YU = BORDER.MAXTOP.Y + THICKNESS; //ВЕРХНЯЯ ГРАНИЦА ОПИСАННОГО
ПРЯМОУГОЛЬНИКА
    INT YL = BORDER.MINTOP.Y - THICKNESS; // НИЖНЯЯ ГРАНИЦА ОПИСАННОГО
ПРЯМОУГОЛЬНИКА
    FOR (INT I = YL; I<YU+1; I++){ //ПРОБЕГ ПО ОПИСАННОМУ
ПРЯМОУГОЛЬНИКУ
        FOR (INT J = XL; J<XR+1; J++){
            IF (I < 0 || J < 0 || I >= H || J >= W) //НА СЛУЧАЙ ВЫХОДА
ЗА ГРАНИЦУ ОБВОДИМ ТУ ЧАСТЬ, КОТОРАЯ ВХОДИТ В РАМКИ ИЗОБРАЖЕНИЯ
                CONTINUE;
            IF (((I > BORDER.MAXTOP.Y) || (I < BORDER.MINTOP.Y)) ||
((J < BORDER.MINTOP.X) || (J > BORDER.MAXTOP.X))){ //ЗАКРАШИВАНИЕ
ТОЛЬКО ТОЙ ОБЛАСТИ, КОТОРАЯ ЛЕЖИТ МЕЖДУ ОПИСАННЫМ И ИСХОДНЫМ
ПРЯМОУГОЛЬНИКОМ
                ARR[I][J] = COLOR_LINE;
            }
        }
    }
}

//COMPLETE
VOID FINDALLRECTANGLE(INT** POSITION, INT H, INT W, RGB** ARR, RGB
COLOR_LINE, INT THICKNESS){
    LONG LONG INT S = 0; //ПЛОЩАДЬ ФИГУРЫ
    FIGURE BORDER; //ГРАНИЦЫ ФИГУРЫ
    INT FLAG_RECTANGLE = 0; //ФЛАЖОК ДЛЯ ОБОЗНАЧЕНИЯ ПРЯМОУГОЛЬНИКА
    INT COUNTER = 0; //ДЛЯ УЧЕТА СЛУЧАЯ, ЧТО НИ ОДИН ПРЯМОУГОЛЬНИК НЕ
НАЙДЕН
    PRINTF("\N");
    PRINTF("КООРДИНАТЫ ЛЕВОГО-ВЕРХНЕГО И ПРАВОГО-НИЖНЕГО УГЛА
НАЙДЕННЫХ ПРЯМОУГОЛЬНИКОВ:\N");
    FOR (INT I = 0; I < H; I++) {
          FOR (INT J = 0; J < W; J++) {
               IF(POSITION[I][J] == 1) { //ЕСЛИ НУЖНАЯ ЗАЛИВКА
                BORDER.TOP1.X = J; BORDER.TOP1.Y = I;
                BORDER.TOP2.X = J; BORDER.TOP2.Y = I;
                BORDER.TOP3.X = J; BORDER.TOP3.Y = I;
                BORDER.TOP4.X = J; BORDER.TOP4.Y = I;
                BORDER.MINTOP.X = J; BORDER.MINTOP.Y = I;
                BORDER.MAXTOP.X = J; BORDER.MAXTOP.Y = I;
//ИНИЦИАЛИЗАЦИЯ ВСЕХ ГРАНИЦ (ДЛЯ ТОЧКИ)
                FILL(I, J, &BORDER, POSITION, H, W); //ПОИСК ВСЕХ
ГРАНИЦ И ВЕРШИН
                S = COUNT_AREA(BORDER, POSITION); //ПОДСЧЕТ ПЛОЩАДИ
                FLAG_RECTANGLE = CHECK_RECTANGLE(BORDER, S);
//ПРОВЕРКА ФИГУРЫ НА ТО, ЧТО ОН ПРЯМОУГОЛЬНИК
                IF (FLAG_RECTANGLE == 1){
                    COUNTER++;
```

```
                    PRINTF("ЛЕВЫЙ-ВЕРХНИЙ:%D %D, ПРАВЫЙ-
НИЖНИЙ:%D %D\N", BORDER.TOP1.X, BORDER.TOP1.Y, BORDER.TOP3.X,
BORDER.TOP3.Y); //ВЫВОД ЛЕВОГО ВЕРХНЕГО, ПРАВОГО НИЖНЕГО УГЛА, ПЛОЩАДИ
ПРЯМОУГОЛЬНИКА
                    HIGHTLIGHTRECTANGLE(ARR, COLOR_LINE, THICKNESS,
BORDER, H, W); //ОБВОДКА ПРЯМОУГОЛЬНИКА
                }
                S = 0; //ЗАНУЛЕНИЕ ПЛОЩАДИ ДЛЯ СЛЕД ФИГУРЫ
                }
            }
        }
    IF (COUNTER == 0){
        PRINTF("ПРЯМОУГОЛЬНИКОВ НЕ НАЙДЕНО\N");
    }
}


// COMPLETE
VOID RGB_FILTER(CHAR* COMPONENT, INT VALUE, INT H, INT W, RGB**
ARR){ // УСТАНАВЛИВАЕТ ЗАДАННУЮ КОМПОНЕНТУ В ЗНАЧЕНИЕ 0 ИЛИ 255
    CHAR C = COMPONENT[0];
    SWITCH(C){
        CASE 'R':
            FOR (INT I = 0; I<H; I++){
                FOR (INT J = 0; J<W; J++){
                    ARR[I][J].R = VALUE;
                }
            }
            BREAK;

        CASE 'G':
            FOR (INT I = 0; I<H; I++){
                FOR (INT J = 0; J<W; J++){
                    ARR[I][J].G = VALUE;
                }
            }
            BREAK;

        CASE 'B':
            FOR (INT I = 0; I<H; I++){
                FOR (INT J = 0; J<W; J++){
                    ARR[I][J].B = VALUE;
                }
            }
            BREAK;
    }
}


// COMPLETE
VOID DRAWCIRCLE(INT X0, INT Y0, INT RADIUS, RGB** ARR, INT H, INT W,
INT THICKNESS, INT FILL, RGB LINE_COLOR, RGB FILL_COLOR){ //РИСОВАНИЕ
ОКРУЖНОСТИ С ЗАПОЛНЕНИЕМ И ТОЛЩИНОЙ
    INT XL = X0 + RADIUS; INT YL = Y0 - RADIUS; // КООРДИНАТЫ ЛЕВОГО
ВЕРХНЕГО УГЛА КВАДРАТА, ОПИСАННОГО ОКОЛО ИСКОМОЙ ОКРУЖНОСТИ
    INT XR = X0 - RADIUS; INT YR = Y0 + RADIUS; // -//- ПРАВЫЙ НИЖНИЙ
УГОЛ
```

```
    FOR(INT I=XR+1; I<XL; I++) {
        FOR(INT J=YL+1; J<YR; J++) {
            IF (I < 0 || J < 0 || I>=H || J>=W)
                CONTINUE;
            INT POINT_1 = (Y0-J)*(Y0-J);
            INT POINT_2 = (X0-I)*(X0-I);
            INT CIRCLE_IN = (RADIUS-THICKNESS) > 0 ? RADIUS-THICKNESS:
0;
            IF ((POINT_1 + POINT_2 <= RADIUS*RADIUS) && (POINT_1 +
POINT_2 >= (CIRCLE_IN*CIRCLE_IN))) {
                ARR[I][J] = LINE_COLOR;
            }
            ELSE IF ((POINT_1 + POINT_2 <= RADIUS*RADIUS) && FILL){
                ARR[I][J] = FILL_COLOR;
            }
        }
    }
}

// COMPLETE
VOID PRE_DRAWCIRCLE(INT XL, INT YL, INT XR, INT YR, RGB** ARR, INT H,
INT W, INT THICKNESS, INT FILL, RGB LINE_COLOR, RGB FILL_COLOR){
    INT X0 = (XL + XR) / 2;
    INT Y0 = (YL + YR) / 2;
    INT RADIUS = YR - Y0;
    DRAWCIRCLE(X0, Y0, RADIUS, ARR, H, W, THICKNESS, FILL, LINE_COLOR,
FILL_COLOR);
}

TYPEDEF STRUCT CHANGEDFILE{
    RGB** ARR;
    BITMAPFILEHEADER* BMFH;
    BITMAPINFOHEADER* BMIF;
} CHANGEDFILE;

//COMPLETE
CHANGEDFILE* DIVIDE(RGB** ARR, BITMAPFILEHEADER* BMFH,
BITMAPINFOHEADER* BMIF, INT THICKNESS, INT N, INT M, RGB COLOR_LINE){
    INT OLDHEIGHT =  BMIF->HEIGHT;
    INT OLDWIDTH =  BMIF->WIDTH;

    BITMAPFILEHEADER* NEWBMFH =  (BITMAPFILEHEADER*)
MALLOC(SIZEOF(BITMAPFILEHEADER));
    BITMAPINFOHEADER* NEWBMIF =  (BITMAPINFOHEADER*)
MALLOC(SIZEOF(BITMAPINFOHEADER));
    NEWBMFH = BMFH;
    NEWBMIF = BMIF;

    NEWBMIF->HEIGHT += THICKNESS*(N - 1);
    NEWBMIF->WIDTH += THICKNESS*(M - 1);
    INT NEWHEIGHT = NEWBMIF->HEIGHT;
    INT NEWWIDTH = NEWBMIF->WIDTH;

    RGB** NEWARR = MALLOC(NEWHEIGHT*SIZEOF(RGB*) + (4 - (NEWHEIGHT *
SIZEOF(RGB)) % 4) % 4);
    FOR (INT I = 0; I < NEWHEIGHT; I++){
```

```
        NEWARR[I] = MALLOC(NEWWIDTH*SIZEOF(RGB) + (4 -
(NEWWIDTH*SIZEOF(RGB)) % 4) % 4);
    }

    FOR (INT I = 0; I < OLDHEIGHT; I++){
        FOR (INT J = 0; J < OLDWIDTH; J++){
            NEWARR[I][J] = ARR[I][J];
        }
    }

    FOR (INT PARTH = 1; PARTH < N; PARTH++){
        FOR (INT I = OLDHEIGHT - 1 + THICKNESS*PARTH; I >
(PARTH*OLDHEIGHT)/N + THICKNESS*(PARTH - 1); I--){
            FOR (INT J = 0; J < OLDWIDTH; J++){
                IF (I+THICKNESS < NEWHEIGHT){
                    NEWARR[I+THICKNESS][J] = NEWARR[I][J];
                }
            }
        }

        FOR (INT I = 0; I<THICKNESS; I++){
            FOR (INT J = 0; J<NEWWIDTH; J++){
                NEWARR[(PARTH*OLDHEIGHT)/N + THICKNESS*(PARTH - 1) + 1
+ I][J] = COLOR_LINE;
            }
        }
    }

    FOR (INT PARTW = 1; PARTW < M; PARTW++){
        FOR (INT J = OLDWIDTH - 1 + THICKNESS*PARTW; J >
(PARTW*OLDWIDTH)/M + THICKNESS*(PARTW-1); J--){
            FOR (INT I = 0; I<NEWHEIGHT; I++){
                IF (J + THICKNESS < NEWWIDTH){
                    NEWARR[I][J + THICKNESS] = NEWARR[I][J];
                }
            }
        }

        FOR (INT J = 0; J < THICKNESS; J++){
            FOR (INT I = 0; I < NEWHEIGHT; I++){
                NEWARR[I][(PARTW*OLDWIDTH)/M + THICKNESS*(PARTW - 1) +
1 + J] = COLOR_LINE;
            }
        }
    }

    CHANGEDFILE* NEWPICTURE = (CHANGEDFILE*)
MALLOC(SIZEOF(CHANGEDFILE));
    NEWPICTURE->ARR = NEWARR;
    NEWPICTURE->BMFH = NEWBMFH;
    NEWPICTURE->BMIF = NEWBMIF;
    RETURN NEWPICTURE;
}

//СЧИТЫВАНИЕ ФАЙЛА
RGB** READFILE(CHAR* NAME_FILE, BITMAPFILEHEADER* BMFH,
BITMAPINFOHEADER* BMIF, INT* ERR){
```

```
    INT R = NO_ERROR;
    FILE* F = FOPEN(NAME_FILE, "RB");
    RGB** ARR;

    IF (F != NULL){
        FREAD(BMFH, 1, SIZEOF(BITMAPFILEHEADER), F);
        FREAD(BMIF, 1, SIZEOF(BITMAPINFOHEADER), F);

        UNSIGNED INT H = BMIF->HEIGHT; // ВЫСОТА ИЗОБРАЖЕНИЯ В
ПИКСЕЛЯХ
        UNSIGNED INT W = BMIF->WIDTH; // -//- ШИРИНА


        IF (BMIF->BITSPERPIXEL != 24 || BMIF->COLORSINCOLORTABLE){
            R = BMP_ERROR;
        }

        IF (!R){
            INT TRASHSIZE = BMFH->PIXELARROFFSET -
SIZEOF(BITMAPFILEHEADER) - SIZEOF(BITMAPINFOHEADER);
            CHAR* TRASH = MALLOC(SIZEOF(CHAR) * TRASHSIZE);
            FREAD(TRASH, 1, TRASHSIZE, F);
            FREE(TRASH);

            ARR = MALLOC(H*SIZEOF(RGB*));
            FOR (INT I = 0; I < H; I++){ // СЧИТЫВАНИЕ КАРТИНКИ
ПОСТРОЧНО И ЕЕ СОХРАНЕНИЕ В ДВУМЕРНЫЙ МАССИВ, ВЫДЕЛЕННЫЙ ВЫШЕ (ДЛЯ
КАЖДОЙ СТРОЧКИ ПИКСЕЛЕЙ КАРТИНКИ)
                ARR[I] = MALLOC(W * SIZEOF(RGB) + (4 -
(W*SIZEOF(RGB))%4)%4);
                FREAD(ARR[I], 1, W * SIZEOF(RGB) + (4 -
(W*SIZEOF(RGB))%4)%4, F);
            }
        }
    }
    ELSE
    {
        R = OPEN_IMAGE_ERROR;
    }

    *ERR = R;

    RETURN ARR;
}

//ВЫВОД ФАЙЛА
VOID WRITEFILE(BITMAPFILEHEADER* BMFH, BITMAPINFOHEADER* BMIF, RGB**
ARR){
    FILE *FF = FOPEN("OUT.BMP", "WB"); // ОТКРЫТИЕ ФАЙЛА НА ЗАПИСЬ
    UNSIGNED INT H = BMIF->HEIGHT;
    UNSIGNED INT W = BMIF->WIDTH;
    FWRITE(BMFH, 1, SIZEOF(BITMAPFILEHEADER), FF);
    FWRITE(BMIF, 1, SIZEOF(BITMAPINFOHEADER), FF);

    INT TRASHSIZE = BMFH->PIXELARROFFSET - SIZEOF(BITMAPFILEHEADER) -
SIZEOF(BITMAPINFOHEADER);
    CHAR* ADDITION = CALLOC(TRASHSIZE, SIZEOF(CHAR));
```

```
    FWRITE(&ADDITION, 1, TRASHSIZE, FF);
    FREE(ADDITION);

    UNSIGNED INT W = W * SIZEOF(RGB) + (4 - (W*SIZEOF(RGB))%4)%4;
    FOR (INT I = 0; I<H; I++){ // ВЫВОД КАРТИНКИ В OUT.BMP
        FWRITE(ARR[I], 1, W, FF);
        FREE(ARR[I]);
    }
    FREE(ARR);
    FCLOSE(FF);
}

//ВЫБОР ЦВЕТА
INT CHOOSECOLOR(RGB* COLOR, CHAR* S_COLOR){
    INT ERR = NO_ERROR;
    RGB BLACK = {0, 0, 0};
    RGB PIRPLE = {255, 0, 128};
    RGB LIGHT_BLUE = {250, 206, 135};
    RGB BLUE = {255,0,0};
    RGB GREEN = {0, 255, 0};
    RGB YELLOW = {0,255,255};
    RGB ORANGE = {0,102,255};
    RGB RED = {0,0,255};
    RGB GRAY = {190,190,190};
    RGB WHITE = {255,255,255};

    IF (!STRCMP(S_COLOR, "BLACK")) *COLOR = BLACK; ELSE
    IF (!STRCMP(S_COLOR, "PIRPLE")) *COLOR = PIRPLE; ELSE
    IF (!STRCMP(S_COLOR, "BLUE")) *COLOR = BLUE; ELSE
    IF (!STRCMP(S_COLOR, "LIGHT_BLUE")) *COLOR = LIGHT_BLUE; ELSE
    IF (!STRCMP(S_COLOR, "GREEN")) *COLOR = GREEN; ELSE
    IF (!STRCMP(S_COLOR, "YELLOW")) *COLOR = YELLOW; ELSE
    IF (!STRCMP(S_COLOR, "ORANGE")) *COLOR = ORANGE; ELSE
    IF (!STRCMP(S_COLOR, "RED")) *COLOR = RED; ELSE
    IF (!STRCMP(S_COLOR, "GRAY")) *COLOR = GRAY; ELSE
    IF (!STRCMP(S_COLOR, "WHITE")) *COLOR = WHITE; ELSE
    ERR = COLOR_ERROR;

    RETURN ERR;
}

//ПОМОЩЬ
VOID PRINTHELP(){
    PRINTF("HELP:\N");
    PRINTF("\T--OPEN <NAME>\T-O: СЧИТАТЬ ИЗОБРАЖЕНИЕ\N");
    PRINTF("\T--CIRCLE\T-C: НАРИСОВАТЬ ОКРУЖНОСТЬ ПО СЛЕДУЮЩИМ
УСЛОВИЯМ:\N");
    PRINTF("\T\T--RADIUS <РАДИУС> <ВЫСОТА X> <ШИРИНА Y> <ТОЛЩИНА>
<ЦВЕТ ЛИНИИ>\T-R: ПО РАДИУСУ И КООРДИНАТАМ\N");
    PRINTF("\T\T--SQUARE <ВЫСОТА X1> <ШИРИНА Y1> (КООРДИНАТЫ ВЕРХНЕГО
ЛЕВОГО УГЛА) <ВЫСОТА X2> <ШИРИНА Y2> (КООРДИНАТЫ НИЖНЕГО ПРАВОГО УГЛА)
<ТОЛЩИНА> <ЦВЕТ ЛИНИИ>\T-Q: ПО ОПИСАННОМУ КВАДРАТУ\N");
    PRINTF("\T\T\TДОПОЛНИТЕЛЬНЫЕ ФЛАГИ ДЛЯ РАБОТЫ С ОКРУЖНОСТЬЮ:\N");
    PRINTF("\T\T\T--FILL <COLOR>\T-F: ВЫБОР ЦВЕТА ЗАЛИВКИ
ОКРУЖНОСТИ\N");
    PRINTF("\T--FILTER <КОМПОНЕНТ RGB> <ЗНАЧЕНИЕ КОМПОНЕНТА>\T-P: RGB-
ФИЛЬТР\N");
```

```
    PRINTF("\T--DIVIDE <КОЛИЧЕСТВО ЧАСТЕЙ ВДОЛЬ ВЫСОТЫ N> <КОЛИЧЕСТВО
ЧАСТЕЙ ВДОЛЬ ШИРИНЫ M> <ТОЛЩИНА> <ЦВЕТ ЛИНИИ>\T-D: РАЗДЕЛЯЕТ
ИЗОБРАЖЕНИЕ ЛИНИЯМИ НА N*M ЧАСТЕЙ\N");
    PRINTF("\T--FINDALLRECTANGLE <ЦВЕТ ЗАЛИВКИ> <ЦВЕТ ОБВОДКИ>
<ТОЛЩИНА ОБВОДКИ>\T-A: ПОИСК ВСЕХ ПРЯМОУГОЛЬНИКОВ ЗАДАННОГО ЦВЕТА И ИХ
ОБВОДКА\N");
    PRINTF("\T--INFORMATION\T-I: ВЫВЕСТИ ИНФОРМАЦИЮ ОБ СЧИТАННОМ
ФАЙЛЕ\N");
    PRINTF("\T--HELP\T-H: ОТКРЫТЬ ПОДСКАЗКУ\N");
    PRINTF("\T\TДОСТУПНЫ СЛЕДУЮЩИЕ ЦВЕТА:\N");
    PRINTF("\T\TBLACK, PIRPLE, BLUE, LIGHT_BLUE, GREEN, YELLOW, ORANGE,
RED, GRAY, WHITE\N");
    PRINTF("\TПРИМЕР РАБОТЫ ПРОГРАММЫ:\N");
    PRINTF("\T\T./EDIT -O YOUR.BMP -C -R 100 300 400 25 PIRPLE\N");
    PRINTF("\T\T./EDIT -O YOUR.BMP -C -R 100 300 400 25 PIRPLE -F
GREEN\N");
    PRINTF("\T\T./EDIT -O YOUR.BMP --CIRCLE --SQUARE 400 200 300 300
100 BLACK -F PIRPLE\N");
    PRINTF("\T\T./EDIT -O YOUR.BMP --CIRCLE --SQUARE 400 200 300 300
25 BLACK --FILL PIRPLE\N");
    PRINTF("\T\T./EDIT -O YOUR.BMP -P R 255\N");
    PRINTF("\T\T./EDIT -O YOUR.BMP --DIVIDE 2 2 25 YELLOW\N");
    PRINTF("\T\T./EDIT --OPEN YOUR.BMP --FINDALLRECTANGLE BLACK RED
25\N");
    PRINTF("\T\T./EDIT -O YOUR.BMP -I\N");
    PRINTF("\T\T./EDIT --HELP\N");
    PRINTF("\TРЕЗУЛЬТАТ СОХРАНЯЕТСЯ В ФАЙЛ OUT.BMP\N");
}

//ПРОВЕРКА КЛЮЧЕЙ
INT ARGCHECKS(CHAR* ARG){
    INT ERR = NO_ERROR;
    IF (!STRCMP(ARG, "--OPEN") || !STRCMP(ARG, "-O")){
        ERR = ARG_ERROR;
    }
    ELSE IF (!STRCMP(ARG, "--CIRCLE") || !STRCMP(ARG, "-C")) {
        ERR = ARG_ERROR;
    }
    ELSE IF (!STRCMP(ARG, "--RADIUS") || !STRCMP(ARG, "-R")) {
        ERR = ARG_ERROR;
    }
    ELSE IF (!STRCMP(ARG, "--SQUARE") || !STRCMP(ARG, "-Q")) {
        ERR = ARG_ERROR;
    }
    ELSE IF (!STRCMP(ARG, "--FILL") || !STRCMP(ARG, "-F")) {
        ERR = ARG_ERROR;
    }
    ELSE IF (!STRCMP(ARG, "--DIVIDE") || !STRCMP(ARG, "-D")) {
        ERR = ARG_ERROR;
    }
    ELSE IF (!STRCMP(ARG, "--FILTER") || !STRCMP(ARG, "-P")) {
        ERR = ARG_ERROR;
    }
    ELSE IF (!STRCMP(ARG, "--FINDALLRECTANGLE") || !STRCMP(ARG, "-
A")){
        ERR = ARG_ERROR;
    }
```

```
    ELSE IF (!STRCMP(ARG, "--INFORMATION") || !STRCMP(ARG, "-I")){
        ERR = ARG_ERROR;
    }
    ELSE IF (!STRCMP(ARG, "--HELP") || !STRCMP(ARG, "-H")) {
        ERR = ARG_ERROR;
    }

    RETURN ERR;
}

//ПРОВЕРКИ ВХОДНЫХ ПАРАМЕТРОВ ФУНКЦИЙ
INT CHECKARGCIRCLERADIUS(INT X1, INT Y1, INT RADIUS, INT THICKNESS,
INT H, INT W){
    INT ERR = NO_ERROR;
    IF (X1 < 0 || X1 >= H) ERR = CIRCLE_RADIUS_ERROR; ELSE
    IF (Y1 < 0 || Y1 >= W) ERR = CIRCLE_RADIUS_ERROR; ELSE
    IF (RADIUS <= 0) ERR = CIRCLE_RADIUS_ERROR; ELSE
    IF (THICKNESS < 1 || THICKNESS >= W || THICKNESS >= H) ERR =
CIRCLE_RADIUS_ERROR;

    RETURN ERR;
}

INT CHECKARGCIRCLESQUARE(INT XL, INT YL, INT XR, INT YR, INT THICKNESS,
INT H, INT W){
    INT ERR = NO_ERROR;
    INT DX = XL - XR; //ВЫСОТЫ
    INT DY = YR - YL; //ШИРОТЫ
    IF (XL < 0 || XL >= H) ERR = CIRCLE_SQUARE_ERROR; ELSE
    IF (YL < 0 || YL >= W) ERR = CIRCLE_SQUARE_ERROR; ELSE
    IF (XR < 0 || XR >= H) ERR = CIRCLE_SQUARE_ERROR; ELSE
    IF (YR < 0 || YR >= W) ERR = CIRCLE_SQUARE_ERROR; ELSE
    IF (DX <= 0 || DY <= 0) ERR = CIRCLE_SQUARE_ERROR; ELSE
    IF (DX != DY) ERR = CIRCLE_SQUARE_ERROR; ELSE
    IF (THICKNESS < 1 || THICKNESS >= W || THICKNESS >= H) ERR =
CIRCLE_SQUARE_ERROR;

    RETURN ERR;
}

INT CHECKARGRGBFILTER(CHAR* COMPONENT, INT VALUE){
    INT ERR = NO_ERROR;
    IF (STRCMP(COMPONENT, "R") && STRCMP(COMPONENT, "G") &&
STRCMP(COMPONENT, "B")) ERR = RGB_FILTER_ERROR; ELSE
    IF (VALUE < 0 || VALUE > 255) ERR = RGB_FILTER_ERROR;

    RETURN ERR;
}

INT CHECKARGDIVIDE(INT N, INT M, INT THICKNESS, INT H, INT W){
    INT ERR = NO_ERROR;
    IF (THICKNESS < 1) ERR = DIVIDE_ERROR; ELSE
    IF (N < 1 || N >= H) ERR = DIVIDE_ERROR; ELSE
    IF (M < 1 || M >= W) ERR = DIVIDE_ERROR;

    RETURN ERR;
}
```

```
INT CHECKARGFINDALLRECTANGLE(INT THICKNESS, INT H, INT W){
    INT ERR = NO_ERROR;
    IF (THICKNESS < 1 || THICKNESS >= W || THICKNESS >= H) ERR =
FIND_ALL_RECTANGLE_ERROR;

    RETURN ERR;
}
//

INT MAIN(INT ARGC, CHAR* ARGV[]){

    INT ERR = NO_ERROR; // ОТВЕЧАЕТ ЗА ОШИБКИ

    //
    BITMAPFILEHEADER* BMFH = (BITMAPFILEHEADER*)
MALLOC(SIZEOF(BITMAPFILEHEADER));
    BITMAPINFOHEADER* BMIF = (BITMAPINFOHEADER*)
MALLOC(SIZEOF(BITMAPINFOHEADER));
    RGB** ARR;
    INT X1 = 0;
    INT X2 = 0;
    INT Y1 = 0;
    INT Y2 = 0;
    INT RADIUS = 0;
    CHAR* COLOR;
    RGB FILL_COLOR = {0, 0, 0};
    RGB LINE_COLOR = {0, 0, 0};
    INT THICKNESS = 1;
    INT N = 1;
    INT M = 1;
    CHAR* COMPONENT;
    INT VALUE = 0;
    CHANGEDFILE* NEWFILEINFO;

    IF (ARGC == 1){
        ERR = KEY_ERROR;
    }

    CHAR* NAME_FILE = (CHAR*) CALLOC (40, SIZEOF(CHAR)); // НАЗВАНИЕ
СЧИТЫВАЕМОГО ФАЙЛА

    //OPTERR = 0;

    CHAR* SHORT_OPTIONS = "O:CR:Q:F:P:D:A:IH?"; //ВСЕ КОРОТКИЕ
ПАРАМЕТРЫ
    INT COUNT = 0; //КОЛИЧЕСТВО
    INT IDX = 0; //ИНДЕКС
    INT WRITE_FLAG = 0; //РАЗРЕШЕНИЕ НА ВЫВОД ФАЙЛА
    INT FILL_FLAG = 0; // ЗАПОЛНЕНИЕ КРУГА
    INT CIRCLE_RADIUS_FLAG = 0; // КРУГ С РАДИУСОМ
    INT CIRCLE_SQUARE_FLAG = 0; // КРУГ С ОПИСАННЫМ КВАДРАТОМ
    INT CHANGE_FILE_FLAG = 0; //ИЗМЕНЕНИЕ В СТРУКТУРЕ САМОГО
ИЗОБРАЖЕНИЯ

    STRUCT OPTION LONG_OPTIONS[] = { // ДЛИННЫЕ ОПЦИИ
        {"OPEN", 1, NULL, 'O'},
```

```
        {"CIRCLE", NO_ARGUMENT, NULL, 'C'},
        {"RADIUS", REQUIRED_ARGUMENT, NULL, 'R'},
        {"SQUARE", REQUIRED_ARGUMENT, NULL, 'Q'},
        {"FILL", REQUIRED_ARGUMENT, NULL, 'F'},
        {"FILTER", REQUIRED_ARGUMENT, NULL, 'P'},
        {"DIVIDE", REQUIRED_ARGUMENT, NULL, 'D'},
        {"FINDALLRECTANGLE", REQUIRED_ARGUMENT, NULL, 'A'},
        {"INFORMATION", NO_ARGUMENT, NULL, 'I'},
        {"HELP", 0, NULL, 'H'},
        {NULL, 0, NULL, 0} //?
    };

    INT OPT = 0; //ТЕКУЩАЯ ОПЦИЯ
    INT LONGINDEX = -1; //ИНДЕКС ОПЦИИ
    WHILE (((OPT = GETOPT_LONG(ARGC, ARGV, SHORT_OPTIONS, LONG_OPTIONS,
&LONGINDEX)) != -1) && !ERR){
        SWITCH (OPT){
            CASE 'O': {
                STRCAT(NAME_FILE, OPTARG);
                ARR = READFILE(NAME_FILE, BMFH, BMIF, &ERR);
                WRITE_FLAG = 1;
                BREAK;
            };
            CASE 'C': {
                IF (WRITE_FLAG == 0){
                    ERR = INFORMATION_ERROR;
                    BREAK;
                }

                OPT = GETOPT_LONG(ARGC, ARGV, SHORT_OPTIONS,
LONG_OPTIONS, &LONGINDEX);
                IF (OPT != -1 && OPT == 'R'){
                    ERR = ARGCHECKS(OPTARG);
                    IF (!ERR){
                        IDX = OPTIND;
                    }

                    IF (!ERR){
                        RADIUS = ATOI(ARGV[IDX-1]);
                    }

                    IF (!ERR && (!ARGCHECKS(ARGV[IDX-1])) && (IDX <
ARGC)){
                        X1 = ATOI(ARGV[IDX]);
                    }
                    ELSE {
                        ERR = ARG_ERROR;
                    }
                    IF (!ERR && (!ARGCHECKS(ARGV[IDX])) && (IDX + 1 <
ARGC)){
                        Y1 = ATOI(ARGV[IDX+1]);
                    }
                    ELSE {
                        ERR = ARG_ERROR;
                    }
```

```
                    IF (!ERR && (!ARGCHECKS(ARGV[IDX+1])) && (IDX + 2
< ARGC)){
                        THICKNESS = ATOI(ARGV[IDX+2]);
                    }
                    ELSE {
                        ERR = ARG_ERROR;
                    }

                    IF (!ERR && (!ARGCHECKS(ARGV[IDX+2])) && (IDX + 3
< ARGC)){
                        COLOR = ARGV[IDX+3];
                        ERR = CHOOSECOLOR(&LINE_COLOR, COLOR);
                    }
                    ELSE {
                        ERR = ARG_ERROR;
                    }

                    IF (!ERR){
                        CIRCLE_RADIUS_FLAG = 1;
                    }
                }
                ELSE IF (OPT != -1 && OPT == 'Q'){
                    ERR = ARGCHECKS(OPTARG);
                    IF (!ERR){
                        IDX = OPTIND;
                    }
                    IF (!ERR){
                        X1 = ATOI(ARGV[IDX-1]);
                    }

                    IF (!ERR && (!ARGCHECKS(ARGV[IDX-1])) && (IDX <
ARGC)){
                        Y1 = ATOI(ARGV[IDX]);
                    }
                    ELSE {
                        ERR = ARG_ERROR;
                    }

                    IF (!ERR && (!ARGCHECKS(ARGV[IDX])) && (IDX + 1 <
ARGC)){
                        X2 = ATOI(ARGV[IDX+1]);
                    }
                    ELSE {
                        ERR = ARG_ERROR;
                    }

                    IF (!ERR && (!ARGCHECKS(ARGV[IDX+1])) && (IDX + 2
< ARGC)){
                        Y2 = ATOI(ARGV[IDX+2]);
                    }
                    ELSE {
                        ERR = ARG_ERROR;
                    }

                    IF (!ERR && (!ARGCHECKS(ARGV[IDX+2])) && (IDX + 3
< ARGC)){
                        THICKNESS = ATOI(ARGV[IDX+3]);
```

```
                    }
                    ELSE {
                        ERR = ARG_ERROR;
                    }

                    IF (!ERR && (!ARGCHECKS(ARGV[IDX+3])) && (IDX + 4
< ARGC)){
                        COLOR = ARGV[IDX+4];
                        ERR = CHOOSECOLOR(&LINE_COLOR, COLOR);
                    }
                    ELSE {
                        ERR = ARG_ERROR;
                    }

                    IF (!ERR){
                        CIRCLE_SQUARE_FLAG = 1;
                    }
                }
                ELSE {
                    ERR = KEY_ERROR;
                }

                IF (!ERR){
                    IF ((OPT = GETOPT_LONG(ARGC, ARGV, SHORT_OPTIONS,
LONG_OPTIONS, &LONGINDEX)) == 'F'){
                        COLOR = OPTARG;
                        ERR = CHOOSECOLOR(&FILL_COLOR, COLOR);
                        IF (!ERR){
                            FILL_FLAG = 1;
                        }
                    }
                }

                IF (!ERR){
                    IF (CIRCLE_RADIUS_FLAG){
                        ERR = CHECKARGCIRCLERADIUS(X1, Y1, RADIUS,
THICKNESS, BMIF->HEIGHT, BMIF->WIDTH);
                        IF (!ERR){
                            DRAWCIRCLE(X1, Y1, RADIUS, ARR, BMIF-
>HEIGHT, BMIF->WIDTH, THICKNESS, FILL_FLAG, LINE_COLOR, FILL_COLOR);
                        }
                    }
                    ELSE IF (CIRCLE_SQUARE_FLAG){
                        ERR = CHECKARGCIRCLESQUARE(X1, Y1, X2, Y2,
THICKNESS, BMIF->HEIGHT, BMIF->WIDTH);
                        IF (!ERR){
                            PRE_DRAWCIRCLE(X1, Y1, X2, Y2, ARR, BMIF-
>HEIGHT, BMIF->WIDTH, THICKNESS, FILL_FLAG, LINE_COLOR, FILL_COLOR);
                        }
                    }
                }

                CIRCLE_RADIUS_FLAG = 0;
                CIRCLE_SQUARE_FLAG = 0;
                FILL_FLAG = 0;
                BREAK;
            };
```

```
            CASE 'P': {
                IF (WRITE_FLAG == 0){
                    ERR = INFORMATION_ERROR;
                    BREAK;
                }

                ERR = ARGCHECKS(OPTARG);
                IF (!ERR){
                    IDX = OPTIND;
                }

                IF (!ERR){
                    COMPONENT = ARGV[IDX-1];
                }

                IF (!ERR && (!ARGCHECKS(ARGV[IDX-1])) && (IDX <
ARGC)){
                    VALUE = ATOI(ARGV[IDX]);
                }
                ELSE {
                    ERR = ARG_ERROR;
                }

                IF (!ERR){
                    ERR = CHECKARGRGBFILTER(COMPONENT, VALUE);
                    IF (!ERR){
                        RGB_FILTER(COMPONENT, VALUE, BMIF->HEIGHT,
BMIF->WIDTH, ARR);
                    }
                }

                BREAK;
            };
            CASE 'D': {
                IF (WRITE_FLAG == 0){
                    ERR = INFORMATION_ERROR;
                    BREAK;
                }

                ERR = ARGCHECKS(OPTARG);
                IF (!ERR){
                    IDX = OPTIND;
                }

                IF (!ERR){
                    N = ATOI(ARGV[IDX-1]);
                }

                IF (!ERR && (!ARGCHECKS(ARGV[IDX-1])) && (IDX <
ARGC)){
                    M = ATOI(ARGV[IDX]);
                }
                ELSE {
                    ERR = ARG_ERROR;
                }
```

```
                IF (!ERR && (!ARGCHECKS(ARGV[IDX])) && (IDX + 1 <
ARGC)){
                    THICKNESS = ATOI(ARGV[IDX+1]);
                }
                ELSE {
                    ERR = ARG_ERROR;
                }

                IF (!ERR && (!ARGCHECKS(ARGV[IDX+1])) && (IDX + 2 <
ARGC)){
                    COLOR = ARGV[IDX+2];
                    ERR = CHOOSECOLOR(&LINE_COLOR, COLOR);
                }
                ELSE {
                    ERR = ARG_ERROR;
                }

                IF (!ERR){
                    ERR = CHECKARGDIVIDE(N, M, THICKNESS, BMIF->HEIGHT,
BMIF->WIDTH);
                    IF (!ERR){
                        NEWFILEINFO = DIVIDE(ARR, BMFH, BMIF,
THICKNESS, N, M, LINE_COLOR);
                        CHANGE_FILE_FLAG = 1;
                        BMIF = NEWFILEINFO->BMIF;
                        BMFH = NEWFILEINFO->BMFH;
                        ARR = NEWFILEINFO->ARR;
                    }
                }

                BREAK;
            };
            CASE 'A': {
                IF (WRITE_FLAG == 0){
                    ERR = INFORMATION_ERROR;
                    BREAK;
                }

                ERR = ARGCHECKS(OPTARG);
                IF (!ERR){
                    IDX = OPTIND;
                }
                IF (!ERR){
                    COLOR = ARGV[IDX-1];
                    ERR = CHOOSECOLOR(&FILL_COLOR, COLOR);
                }

                IF (!ERR && (!ARGCHECKS(ARGV[IDX-1])) && (IDX <
ARGC)){
                    COLOR = ARGV[IDX];
                    ERR = CHOOSECOLOR(&LINE_COLOR, COLOR);
                }
                ELSE {
                    ERR = ARG_ERROR;
                }
```

```
                IF (!ERR && (!ARGCHECKS(ARGV[IDX])) && (IDX + 1 <
ARGC)){
                    THICKNESS = ATOI(ARGV[IDX+1]);
                }
                ELSE {
                    ERR = ARG_ERROR;
                }

                IF (!ERR){
                    ERR = CHECKARGFINDALLRECTANGLE(THICKNESS, BMIF-
>HEIGHT, BMIF->WIDTH);
                    IF (!ERR){
                        INT** POSITION = (INT**) CALLOC(BMIF->HEIGHT,
SIZEOF(INT*));
                        FOR (INT I = 0; I<BMIF->HEIGHT; I++){
                            POSITION[I] = (INT*) CALLOC(BMIF->WIDTH,
SIZEOF(INT));
                        }
                        CREATEARRPOSITION(ARR, FILL_COLOR, BMIF-
>HEIGHT, BMIF->WIDTH, POSITION);
                        FINDALLRECTANGLE(POSITION, BMIF->HEIGHT, BMIF-
>WIDTH, ARR, LINE_COLOR, THICKNESS);
                        FOR (INT I = 0; I<BMIF->HEIGHT; I++){
                            FREE(POSITION[I]);
                        }
                        FREE(POSITION);
                    }
                }

                BREAK;
            };
            CASE 'I':{
                IF (ERR != 2 && WRITE_FLAG == 1){
                    PRINTFILEHEADER(*BMFH);
                    PRINTINFOHEADER(*BMIF);
                }
                ELSE {
                    ERR = INFORMATION_ERROR;
                }

                BREAK;
            };
            CASE 'F': {
                ERR = FLAG_BEFORE_ERROR;
                BREAK;
            };
            CASE 'R': {
                ERR = FLAG_BEFORE_ERROR;
                BREAK;
            }
            CASE 'Q':{
                ERR = FLAG_BEFORE_ERROR;
                BREAK;
            }
            CASE '?':
                ERR = KEY_ERROR;
                BREAK;
```

```
            CASE 'H':
                PRINTHELP();
                BREAK;
        }
    }


    //ВЫВОД ИЗМЕНЕННОГО ФАЙЛА
    IF (WRITE_FLAG && !ERR){
        WRITEFILE(BMFH, BMIF, ARR);
    }

    //ВЫВОД ОШИБОК
    IF (ERR){
        PRINTHELP();
        FPRINTF(STDERR, "ОШИБКА: %S.\N", ERROR_MSG[ERR]);
    }

    // ОЧИСТКА ПАМЯТИ
    FREE(NAME_FILE);
    IF (NEWFILEINFO)
        FREE(NEWFILEINFO);
    FREE(BMFH);
    FREE(BMIF);
    RETURN 0;
}
```