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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

ТЕМА: РАБОТА С BMP ФАЙЛАМИ

Студент гр. 6304 _____ Ковынев М.В.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2017

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Ковынев М.В.

Группа 6304

Тема работы: **РАБОТА С BMP ФАЙЛАМИ**

Содержание пояснительной записки:

- Содержание
- Введение
- Описание функций
- Примеры работы программы
- Заключение
- Список использованных источников
- Приложение: Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: _____

Дата сдачи реферата: _____

Дата защиты реферата: _____

Студент _____ Ковынев М.В.

Преподаватель _____ Берленко Т.А.

Аннотация

В данной работе был создан проект на языке программирования C, который позволяет генерировать BMP изображения и текстовые данные и проверять, корректно ли работает пользовательская программа.

Для функционирования проекта были созданы:

1. Генератор тестов и BMP изображений
2. Эталонное решение поставленной задачи
3. Скрипт, который вызывает вышеназванные программы и сверяет результаты эталонного решения с пользовательским решением.

Были созданы и описаны необходимые функции, позволяющие считывать и изменять данные генерируемых BMP изображений, а также создавать новые. Помимо этого, была проведена работа над оптимизацией исходного кода программы для ускорения ее быстродействия и оптимального использования памяти и ресурсов. Приведено полное описание исходного кода и примера работы программы.

Содержание

Введение

Цель работы

Создание проекта, позволяющего проверить, корректно ли работает решение поставленной задачи; генерация и обработка файлов в формате BMP. Написание программы генератора тестов, эталонного решения задачи и скрипта, связывающих эти программы с проверяемой, оценка ее работы.

Формулировка задачи

Требуется создать проект, содержащий следующие программы:

- *Пользовательское решение* - решение задачи, которое предоставил пользователь (исходный код).
- *Эталонное решение* - корректное решение задачи (исходный код).
- *Чекер* - набор файлов с исходным кодом и скрипт, написанный на bash. Последний сравнивает результаты эталонного и пользовательского решений и дает однозначный ответ, верно ли пользователь решил задачу.

Содержимое checker

- `generate.c` - Генерация тестовых данных.
- `refsol.c` - Эталонное решение задачи.
- `Makefile` - Удаление объектных и исполняемых файлов, компиляция файлов с исходным кодом.
- `scriprt.sh`
    - запуск генерации тестовых данных
    - запуск эталонного решения с данным набором тестовых данных
    - запуск пользовательского решения с данным набором тестовых данных
    - сравнение результатов и вывод ответа, верное ли решение

Индивидуальное задание

Общая постановка задачи: обработка файла в формате BMP.

Требуется реализовать программу, которая:

1. находит в файле самый большой белый прямоугольник и выводит координаты левого верхнего угла и правого нижнего.
2. рисует прямоугольный треугольник в заданной области черным цветом. Прямой угол должен лежать в левом нижнем углу области, длина катетов определяется длиной стороны области, к которой они прилегают, и равна половине этой стороны.
3. отражает область в файле по горизонтали.
4. сохраняет результат в новом файле.

(число здесь идентифицирует номер команды)

Параметры

Программа получает параметры из входного потока и должна проверить их корректность. Параметры:

- *input_file*
- *x0*
- *y0*
- *x1*
- *y1*
- *commands*
- *input* - *BMP* файл
- *x0 y0* - левый верхний угол области (отсчет с точки 0, 0)
- *x1 y1* - правый нижний угол области
- *commands* - числовой массив неизвестной длины, который хранит в себе последовательность функций обработки входного файла. Массив заканчивается числом 4 - функцией сохранения результата в новом файле.

В случае, если программа получила некорректные параметры, то:

- операционной системе возвращается ненулевой код возврата (*return* в *main*)
- не создается выходного файла
- выводится сообщение об ошибке “*Fail with* <имя параметра>”.

Ход работы

1. Генератор данных

1.1. Структура BMP файла

Любое BMP изображение устроено следующим образом: (см. рис. 1)

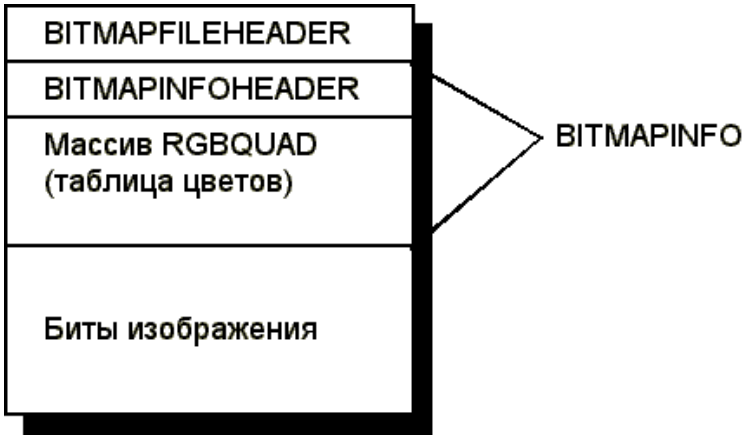

Рисунок 1. Структура BMP файла

В начале стоит заголовок файла (`BITMAPFILEHEADER`). Он описан следующим образом:

```
/*=====================================================================
  Структура BITMAPFILEHEADER, содержащая
  информацию о типе, размере и макете файла
*/
typedef struct
{
	short 		bfType;		// Тип файла (должен быть BM)
	unsigned int 	bfSize;		// Размер файла
	unsigned short 	bfReserved1;	// Зарезервирован (должен быть 0)
	unsigned short 	bfReserved2;	// Зарезервирован (должен быть 0)
	unsigned int 	bfOffBits;	// Смещение от начала битов

} BITMAPFILEHEADER;
//=====================================================================
```

После структуры `BITMAPFILEHEADER` идет структура `BITMAPINFOHEADER`, которая объявлена так:

```
/*=====================================================================
  Структура BITMAPINFOHEADER, содержащая
  информацию о размерах и цветовом формате
*/
```

```
typedef struct
{
	unsigned int    biSize;          // Число байтов необходимое для структуры
	int             biWidth;         // Ширина изображения (в пикселях)
	int             biHeight;        // Высота изображения (в пикселях)
	unsigned short  biPlanes;        // Число плоскостей (должен быть 1)
	unsigned short  biBitCount;      // Число битов на пиксель (24 бита)
	unsigned int    biCompression;   // Тип сжатия
	unsigned int    biSizeImage;     // Размер изображения (в байтах)
	int             biXPelsPerMeter; // Разрешающая способность по горизонтали
	int             biYPelsPerMeter; // Разрешающая способность по вертикали
	unsigned int    biClrUsed;       // Количество используемых цветов
	unsigned int    biClrImportant;  // Количество существенных цветов

} BITMAPINFOHEADER;
//=====================================================================
```

Учитывая, что необходимо генерировать 24 битные изображения, тогда у данных изображений нет палитры (беспалитровый BMP), то есть цвет высчитывается прямо из тех битов, которые идут в файле.

Для этого реализуем структуру `RGBPIXEL`, описывающую один пиксель BMP файла.

```
/*=====================================================================
  Структура RGBPIXEL, содержащая информацию о пикселе изображения
*/
typedef struct
{
	unsigned char  rgbBlue;   // Синий   цвет (от 0 до 255)
	unsigned char  rgbGreen;  // Зеленый цвет (от 0 до 255)
	unsigned char  rgbRed;    // Красный цвет (от 0 до 255)

} RGBPIXEL;
//=====================================================================
```

Структура `BITMAPFILEHEADER` должна занимать 14 байт, а `BITMAPINFOHEADER` – 40 байт, но из-за выравнивания данные структуры занимают большее количество байт. Для решения данной проблемы требуется отменить выравнивание с помощью директивы препроцессора:

```
#pragma pack(push)
#pragma pack(2)
```

1.2. Функция `writeToBitmapHeader`

Для записи структуры `BITMAPFILEHEADER` в BMP файла был создана функция `writeToBitmapHeader`, которая получает в качестве аргумента файл и заголовочную структуру.

```
/*=========================================================================
  Функция WriteToBitmapHeader записывает в файл outputFile
  структуру данных headerBitmap
*/
void writeToBitmapHeader(FILE* outfile, BITMAPFILEHEADER headerBMP);
//=========================================================================

void writeToBitmapHeader(FILE* outputFile, BITMAPFILEHEADER headerBitmap)
{
	fwrite(&headerBitmap, sizeof(BITMAPFILEHEADER), 1, outputFile);
}
```

1.3. Функция `writeToBitmapInfo`

Для записи структуры `BITMAPINFOHEADER` в BMP файл была создана функция `writeToBitmapInfo`, которая получает в качестве аргумента файл и структуру `BITMAPINFOHEADER`.

```
/*=========================================================================
  Функция WriteToBitmapInfo записывает в файл outputFile
  структуру данных infoBitmap
*/
void writeToBitmapInfo(FILE* outfile, BITMAPINFOHEADER infoBMP);
//=========================================================================
void writeToBitmapInfo(FILE* outputFile, BITMAPINFOHEADER infoBitmap)
{
	fwrite(&infoBitmap, sizeof(BITMAPINFOHEADER), 1, outputFile);
}
```

1.4. Функция `writeToBitmapRGB`

Для записи двумерного массива структур пикселей изображения была создана функция `writeToBitmapRGB`.

Циклически, в зависимости от размеров изображения, полученных из структуры `BITMAPINFOHEADER`, записывается каждая структура, отвечающая за текущий пиксель на каждой итерации цикла.

В конце каждой строки пикселей приписываются «мусорные данные» - дополнительно несколько нулей. Количество `extraBytes` зависит от размеров исходного изображения и вычисляется по формуле:

```
int extraBytes = 4 - ((infoBitmap.biWidth * 3) % 4);
	if (extraBytes == 4)
	{
		extraBytes = 0;
	}
```

Функция `writeToBitmapRGB` имеет следующий вид:

```
/*=====================================================================
  Функция WriteToBitmapRGB записывает в файл outputFile матрицу пикселей
  arrayRGB, основываясь на размеры изображения из структуры infoBitmap
*/
void writeToBitmapRGB(FILE* outfile, RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBMP);
//=====================================================================

void writeToBitmapRGB(FILE* outputFile, RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap)
{

	int extraBytes = 4 - ((infoBitmap.biWidth * 3) % 4);
	if (extraBytes == 4)
	{
		extraBytes = 0;
	}

	for (int i = infoBitmap.biHeight - 1; i >= 0; i--)
	{
		for (int j = 0; j < infoBitmap.biWidth; j++)
		{
			fwrite(&arrayRGB[i][j].rgbBlue, 1, 1, outputFile);
			fwrite(&arrayRGB[i][j].rgbGreen, 1, 1, outputFile);
			fwrite(&arrayRGB[i][j].rgbRed, 1, 1, outputFile);
		}

		if (extraBytes)
		{
			for (int j = 0; j < extraBytes; j++)
			{
				fwrite("0", 1, 1, outputFile);
			}
		}
	}

}
```

1.5. Функция `readFromBitmapHeader`

Функция `ReadFromBitmapHeader` считывает из файла `inputFile` структуру `headerBitmap` и возвращает ее

```
BITMAPFILEHEADER readFromBitmapHeader(FILE* inputFile)
{
	BITMAPFILEHEADER headerBitmap;
	fread(&headerBitmap, sizeof(BITMAPFILEHEADER), 1, inputFile);
	return headerBitmap;
}
```

1.6. Функция `readFromBitmapInfo`

Функция `readFromBitmapInfo` считывает из файла `inputFile` структуру `infoBitmap` и возвращает ее

```
BITMAPINFOHEADER readFromBitmapInfo(FILE* inputFile)
{
	BITMAPINFOHEADER infoBitmap;
	fread(&infoBitmap, sizeof(BITMAPINFOHEADER), 1, inputFile);
	return infoBitmap;
}
```

1.7. Функция `readFromBitmapRGB`

Функция `ReadFromBitmapRGB` считывает из файла матрицу пикселей с учетом размеров изображения и возвращает ее.

В конце каждой строки пикселей считываются «мусорные данные». Количество `extraBytes` зависит от размеров исходного изображения и вычисляется по формуле

```
RGBPIXEL** readFromBitmapRGB(FILE* inputFile, BITMAPINFOHEADER infoBitmap)
{
	RGBPIXEL** arrayRGB = (RGBPIXEL**)malloc(sizeof(RGBPIXEL*)*infoBitmap.biHeight);
	for (int i = 0; i < infoBitmap.biHeight; i++)
	{
		arrayRGB[i] = (RGBPIXEL*)malloc(sizeof(RGBPIXEL)*infoBitmap.biWidth);
	}

	int extraBytes = 4 - ((infoBitmap.biWidth * 3) % 4);
	if (extraBytes == 4)
	{
		extraBytes = 0;
	}

	for (int i = infoBitmap.biHeight - 1; i >= 0; i--)
	{
		for (int j = 0; j < infoBitmap.biWidth; j++)
		{
			fread(&arrayRGB[i][j].rgbBlue, 1, 1, inputFile);
			fread(&arrayRGB[i][j].rgbGreen, 1, 1, inputFile);
			fread(&arrayRGB[i][j].rgbRed, 1, 1, inputFile);
		}

		if (extraBytes)
		{
			for (int j = 0; j < extraBytes; j++)
			{
				char zeroByte;
				fread(&zeroByte, 1, 1, inputFile);
			}
		}
	}

	return arrayRGB;
}
```

1.8. Функция `createBitmapFile`

Для создания одного BMP изображения была реализована функция `createBitmapFile`, имеющая следующий прототип:

```
void createBitmapFile(char* fileNameIndex, int desiredColorRed,
             int desiredColorGreen, int desiredColorBlue)
```

- `Width` – ширина изображения
- `Height` – высота изображения
- `fileNameIndex` – номер изображения, используемый для записи имени файла
- `desiredColorRed`, `desiredColorRed`, `desiredColorBlue` – цвет в формате RGB, который будет встречаться в генерируемом изображении чаще других. Количество таких пикселей – не более 33% от всего количества в исходном изображении.

Первоначально в данной функции генерируется имя файла. Оно имеет вид: `"./input_X.bmp"`, где X – номер изображения (`fileNameIndex`). Номер изображения необходим для того, чтобы создать несколько разных BMP изображений и не перезаписать уже созданный файл.

После этого создается файл и заполняются данными структуры `BITMAPFILEHEADER` и `BITMAPINFOHEADER` и выделяется память под двумерный массив структур пикселей.

Для того чтобы заполнить массив BMP файла RGB цветом используются функции `addColorToBitmap` и `addRectangleToBitmap`, описание которых будет представлено далее.

После создания массива цветов необходимо записать сгенерируемые данные в файл `./input_X.bmp`, используя реализованные функции `writeToBitmapHeader`, `writeToBitmapInfo`, и `writeToBitmapRGB`.

Функция `createBitmapFile` имеет следующий вид:

```
void createBitmapFile(char* fileNameIndex, int desiredColorRed,
      int desiredColorGreen, int desiredColorBlue)
{
      char fileName[30] = "./input_";   // Создание имени изображения
      strcat(fileName, fileNameIndex);  // с учетом его номера
      strcat(fileName, ".bmp");         //

      FILE* outfileBMP = fopen(fileName, "wb");

      BITMAPFILEHEADER headerBMP;
      BITMAPINFOHEADER infoBMP;

      // Количество байтов необходимых для выравнивания изображения
      int extraBytes = 4 - ((WIDTH * 3) % 4);
```

```
	if (extraBytes == 4)
	{
		extraBytes = 0;
	}

	// Полный размер изображения с учетом байтов для выравнивания
	int paddedSize = ((WIDTH * 3) + extraBytes) * HEIGHT;

	// Заполнение структур BMP изображения
	headerBMP.bfType = TYPE;
	headerBMP.bfSize = SIZEINFO + SIZEHEADER + paddedSize;
	headerBMP.bfReserved1 = 0;
	headerBMP.bfReserved2 = 0;
	headerBMP.bfOffBits = SIZEINFO + SIZEHEADER;

	infoBMP.biSize = SIZEINFO;
	infoBMP.biWidth = WIDTH;
	infoBMP.biHeight = HEIGHT;
	infoBMP.biPlanes = 1;
	infoBMP.biBitCount = BITCOUNT;
	infoBMP.biCompression = 0;
	infoBMP.biSizeImage = paddedSize;
	infoBMP.biXPelsPerMeter = 0;
	infoBMP.biYPelsPerMeter = 0;
	infoBMP.biClrUsed = 0;
	infoBMP.biClrImportant = 0;

	// Инициализация двумерного массива структур
	RGBPIXEL** arrayRGB = (RGBPIXEL**)malloc(sizeof(RGBPIXEL*)*HEIGHT);
	for (int i = 0; i < HEIGHT; i++)
	{
		arrayRGB[i] = (RGBPIXEL*)malloc(sizeof(RGBPIXEL)*WIDTH);
	}

	// Вызов функции addColorToBitmap
	addColorToBitmap(arrayRGB, infoBMP, desiredColorRed,
		desiredColorGreen, desiredColorBlue);

	// Вызов функций addRectangleToBitmap
	addRectangleToBitmap(arrayRGB, infoBMP, 15, 255, 255, 255);
	addRectangleToBitmap(arrayRGB, infoBMP, 15, desiredColorRed,
		desiredColorGreen, desiredColorBlue);

	// Запись данных BMP изображения
	writeToBitmapHeader(outfileBMP, headerBMP);
	writeToBitmapInfo(outfileBMP, infoBMP);
	writeToBitmapRGB(outfileBMP, arrayRGB, infoBMP);

	// Освобождение памяти
	for (int i = 0; i < HEIGHT; i++)
	{
		free(arrayRGB[i]);
	}

	free(arrayRGB);

	fclose(outfileBMP);
}
```

1.9. Функция `addColorToBitmap`

Функция `addColorToBitmap` получает в качестве аргумента двумерный массив структур `RGBPIXEL`, отвечающий за пиксели BMP изображения и записывает в него коды цветов.

В данной программе реализовано 3 варианта цветов пикселя BMP изображения:

1. Белый цвет

```
arrayRGB[i][j].rgbBlue = 255;   // Заполнение
arrayRGB[i][j].rgbGreen = 255;  // пикселя белым
arrayRGB[i][j].rgbRed = 255;    // цветом
```

2. Произвольный цвет

```
arrayRGB[i][j].rgbBlue = rand() % 256;    // Заполнение
arrayRGB[i][j].rgbGreen = rand() % 256;   // пикселя рандомным
arrayRGB[i][j].rgbRed = rand() % 256;     // цветом
```

3. Переданный цвет

```
arrayRGB[i][j].rgbBlue = desiredColorBlue;    // Заполнение
arrayRGB[i][j].rgbGreen = desiredColorGreen;  // пикселя переданным
arrayRGB[i][j].rgbRed = desiredColorRed;      // цветом
```

Присваивание определенного цвета (белый, произвольный или переданный) зависит от параметра `colorFrequency`, значения которого варьируется от 1 до 3 соответственно.

Функция `addColorToBitmap` имеет следующий вид:

```
void addColorToBitmap(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBMP,
	int desiredColorRed, int desiredColorGreen, int desiredColorBlue)
{

	for (int i = 0; i < infoBMP.biHeight; i++)
	{
		for (int j = 0; j < infoBMP.biWidth; j++)
		{

			// Переменная, отвечающая за цвет пикселя
			// 0 - белый, 1 - рандомный, 2 - переданный
			int colorFrequency;
			colorFrequency = rand() % 3;

			if (colorFrequency == 0)
			{
				arrayRGB[i][j].rgbBlue = 255;   // Заполнение
				arrayRGB[i][j].rgbGreen = 255;  // пикселя белым
				arrayRGB[i][j].rgbRed = 255;    // цветом
			}

			else if (colorFrequency == 1)
```

```
                {
                        arrayRGB[i][j].rgbBlue = rand() % 256;     // Заполнение
                        arrayRGB[i][j].rgbGreen = rand() % 256;    // пикселя рандомным
                        arrayRGB[i][j].rgbRed = rand() % 256;      // цветом
                }

                else
                {
                   arrayRGB[i][j].rgbBlue = desiredColorBlue;     // Заполнение
                   arrayRGB[i][j].rgbGreen = desiredColorGreen;   // пикселя переданным
                   arrayRGB[i][j].rgbRed = desiredColorRed;       // цветом
                }

        }
    }

}
```

1.10. Функция `addRectangleToBitmap`

Функция `addRectangleToBitmap`, добавляет в переданный двумерный массив структур `RGBPIXEL` `numberOfRectangles`. прямоугольников, имеющих цвет `desiredColorRed`, `desiredColorGreen` и `desiredColorBlue`.

Размеры данных прямоугольников прямо пропорциональны размерам исходного изображения и являются динамическими, то есть при увеличении размеров изображений увеличиваются и размеры самих прямоугольников.

Максимальное возможное количество пикселей, занимаемое одним прямоугольников, рассчитывается по формуле: *Ширина* x *Высота / 225*.

Например, для изображения размером 640x360 максимально возможное количество пикселей, занимаемое одним прямоугольников – 1024 пикселя.

Функция `addRectangleToBitmap` имеет следующий вид:

```
void addRectangleToBitmap(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBMP,
    int numberOfRectangles, int desiredColorRed, int desiredColorGreen,
    int desiredColorBlue)
{

    for (int k = 0; k < numberOfRectangles; k++)
    {
        // Инициализация размеров добавляемых прямоуголников
        int RectangleX0Coordinates = rand() % (infoBMP.biWidth * 13 / 15 - 1);
        int RectangleY0Coordinates = rand() % (infoBMP.biHeight * 13 / 15 - 1);
        int RectangleWidth = rand() % (infoBMP.biWidth / 10) + 1;
        int RectangleHeight = rand() % (infoBMP.biHeight / 10) + 1;

        for (int i = RectangleY0Coordinates;
            i < RectangleY0Coordinates + RectangleHeight; i++)
        {
            for (int j = RectangleX0Coordinates;
                j < RectangleX0Coordinates + RectangleWidth; j++)
            {
               arrayRGB[i][j].rgbBlue = desiredColorBlue;     // Заполнение
               arrayRGB[i][j].rgbGreen = desiredColorGreen;   // пикселя переданным
               arrayRGB[i][j].rgbRed = desiredColorRed;       // цветом
```

```
                }
        }
}

}
```

1.11. Функция `createTxtFile`

Функция `createTxtFile` создает текстовый файл с командами. Команды могут быть правильными или неправильными. Корректность команд зависит от параметра `isCorrectCommand`, который может принимать значения: 1 – правильный команды или 0 – неправильные команды.

Учитывая, что программа должна генерировать и некорректные входные данные, то есть шанс, что все команды будут неправильное, поэтому в начале в начале файла записывается строка: `./input_1.bmp 10 50 20 30 1 2 3 3 2 1 4\n` – тем самым среди команд будет хотя бы одна правильная

Функция `createTxtFile` имеет следующий вид:

```
void createTxtFile()
{
	FILE* outfileTXT = fopen("./input.txt", "w");

	// Запись правильных команд первого теста
	fprintf(outfileTXT, "./input_1.bmp 10 50 20 30 1 2 3 3 2 1 4\n");

	for (int i = 1; i < 50; i++)
	{
		// Переменная, отвечающая за количество команд
		int commandsSize = rand() % 12 + 1;


		// Запись пути BMP изображения
		fprintf(outfileTXT, "%s", "./input_");

		// Переменная isCorrectCommand отвечает за
		// корректность генерируемых команд
		// 0 - некорректные, 1 - корректные
		int isCorrectCommand = rand() % 2;


		if (isCorrectCommand)
		{
			// Запись правильного индекса изображения
			fprintf(outfileTXT, "%d.bmp ", rand() % 5 + 1);
		}
		else
		{
			// Запись неправильного индекса изображения
			fprintf(outfileTXT, "%d.bmp ", rand() % 10 + 1);
		}


		isCorrectCommand = rand() % 2;
		if (!isCorrectCommand)
```

```
		{
			// Запись неправильных координат области изображения
		  fprintf(outfileTXT, "%d %d ", rand() % 5000 - 1000, rand() % 5000 - 1000);
		  fprintf(outfileTXT, "%d %d ", rand() % 5000 - 1000, rand() % 5000 - 1000);
		}
		else
		{
			// Генерация корректных команд

			int x0 = 1;
			int y0 = 0;
			int x1 = 0;
			int y1 = 1;

			// Генерация коррекнтых координат по X
			// X0 должен быть больше, чем X1
			while (x0 >= x1)
			{
				x0 = rand() % WIDTH;
				x1 = rand() % HEIGHT;
			}

			// Генерация коррекнтых координат по Y
			// Y1 должен быть больше, чем Y0
			while (y1 >= y0)
			{
				y0 = rand() % WIDTH;
				y1 = rand() % HEIGHT;
			}

			// Запись правильных координат области изображения
			fprintf(outfileTXT, "%d %d ", x0, y0);
			fprintf(outfileTXT, "%d %d ", x1, y1);
		}


		isCorrectCommand = rand() % 2;
		if (!isCorrectCommand)
		{
			// Запись неправильных команд
			for (int i = 0; i < commandsSize; i++)
			{
				fprintf(outfileTXT, "%d ", rand() % 7);
			}

			fprintf(outfileTXT, "\n");
		}
		else
		{
			// Запись правильных команд
			for (int i = 0; i < commandsSize; i++)
			{
				fprintf(outfileTXT, "%d ", rand() % 3 + 1);
			}

			fprintf(outfileTXT, "%d\n", 4);
		}
	}

	fclose(outfileTXT);
}
```

1.12. Функция `main`

Объявим несколько констант, необходимых для работы программы:

```
#define WIDTH      640        // Ширина изображения
#define HEIGHT     360        // Высота изображения
#define TYPE       0x4D42     // Тип файла (должен быть BM)
#define SIZEINFO   40         // Размер структуры BITMAPINFOHEADER
#define SIZEHEADER 14         // Размер структуры BITMAPFILEHEADER
#define BITCOUNT   24         // Число битов на пиксель (24 бита)
```

В главной функции `main` циклически создаются несколько BMP изображений. Номер итерации цикла – номер соответствующего BMP изображения. После создания изображений, создается файл с командами. Функция `main` имеет следующий вид:

```
int main()
{
	srand(time(NULL));

	unsigned int desiredColorRed, desiredColorGreen, desiredColorBlue;

	char fileNameIndex[3];  // Номер изображения

	for (int i = 1; i <= 5; i++)
	{
		// Преобразование номера изображения в строку
		sprintf(fileNameIndex, "%d", i);

		desiredColorRed = rand() % 255;     //
		desiredColorGreen = rand() % 255;   // Генерация цвета фона
		desiredColorBlue = rand() % 255;    //

		// Создание изображения
		createBitmapFile(fileNameIndex, desiredColorRed, desiredColorGreen,
			desiredColorBlue);
	}

	// Создание файла с командами
	createTxtFile();

	return 0;
}
```

1.13. `Makefile` для генератора

```
generate: struct.h generate.o rwBitmap.o
	gcc generate.o -o  generate_main.out rwBitmap.o
	rm *.o
generate.o: generate.c struct.h
	gcc -c generate.c
rwBitmap.o: rwBitmap.c rwBitmap.h struct.h
	gcc -c rwBitmap.c
```

2. Решение задачи

2.1. Функция `mirrorReflectionBitmap`

Для реализации горизонтального отображения изображения была написана функция `mirrorReflectionBitmap`, которая циклически меняет местами симметричные элементы строки BMP файла.

```
void mirrorReflectionBitmap(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap)
{

	RGBPIXEL temp;  // Переменная для обмена пикселями

	for (int i = 0; i < infoBitmap.biHeight; i++)
	{
		for (int j = 0; j < infoBitmap.biWidth / 2; j++)
		{
			// Обмениваем симметричные элементы местами
			temp = arrayRGB[i][j];
			arrayRGB[i][j] = arrayRGB[i][infoBitmap.biWidth - 1 - j];
			arrayRGB[i][infoBitmap.biWidth - 1 - j] = temp;
		}
	}

}
```

2.2. Функция `drawBlackTriangle`

Функция `drawBlackTriangle` рисует в заданной области *x0, y0, x1, y1* прямоугольный треугольник в заданной области черным цветом. Прямой угол лежит в левом нижнем углу области, длина катетов определяется длиной стороны области, к которой они прилегают, и равна половине этой стороны.

Так как данная область может быть не квадратной, то необходимо рассчитать данные для построения треугольника. Треугольник является совокупностью нескольких строк изображения по вертикали, длины которых постепенно увеличиваются от нуля до половины длины катета, прилежащего к оси X.

```
void drawBlackTriangle(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap,
	int x0, int y0, int x1, int y1)
{

	// Массив steps содержит длины отрезков i - строки треугольника
	// От 0 до половины стороны катета, прилежащего к оси X
	float* steps = (float*)malloc(sizeof(float)*(y0 - y1));
	steps[0] = 0.0;

	// Расчет длин отрезков
	for (int i = 1; i < y0 - y1; i++)
	{
```

```
			steps[i] = (float)(x1 - x0) / (float)(y0 - y1) + steps[i - 1];
		}

		int k = 0;
		for (int i = (y1 + y0) / 2; i < y0; i++)
		{
			for (int j = x0; j < x0 + (int)steps[k]; j++)
			{
				arrayRGB[i][j].rgbBlue = 0;		// Заполнение
				arrayRGB[i][j].rgbRed = 0;		// пикселя черным
				arrayRGB[i][j].rgbGreen = 0;		// цветом
			}
			k++;
		}

}
```

2.3. Структура RECTANGLECOORDINATES

Структура RECTANGLECOORDINATES описывает параметры найденной прямоугольной белой области: координаты по оси X, Y и площадь.

```
/*=========================================================================
Структура RECTANGLECOORDINATES описывает параметры найденной
прямоугольной белой области: координаты по оси X,Y и площадь
*/
typedef struct
{
	int x0;
	int y0;
	int x1;
	int y1;
	int area;
} RECTANGLECOORDINATES;
//=========================================================================
```

2.4. Функция searchWhiteRectangle

Функция searchWhiteRectangle самый большой белый прямоугольник и выводит координаты левого верхнего угла и правого нижнего.

Для реализации поиска введем вспомогательный двумерный массив соразмерный с данным изображением, в ячейках которого будем хранить 1, если это белый пиксель и 0, если не белый.

Идея алгоритма состоит в следующем: последовательно для каждой строки изображения создается гистограмма на основе одномерного массива такой же длины, что и изображение.

Если в i-той строке на j-той позиции встречается 0, тогда i,j элемент гистограммы – 0. Если 1, то прибавляем ее к текущему значению i,j элемента гистограммы.

Получаем гистограмму, в которой необходимо найти наибольшую прямоугольную площадь. Для этого реализуем функцию maxHistogramArea,

которая будет возвращать структуру RECTANGLECOORDINATES, в которой будут хранится параметры данной области

```
void searchWhiteRectangle(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap)
{

	// Создаем вспомогательный двумерный массив с такими же размерами, что и изображение
	int** addArrayForRGB = (int**)malloc(sizeof(int**)*infoBitmap.biHeight);
	for (int i = 0; i < infoBitmap.biHeight; i++)
	{
		addArrayForRGB[i] = (int*)malloc(sizeof(int)*infoBitmap.biWidth);
	}


	// Заполняем его следующим образом: 1 - если пиксель белый
	//                                                    0 - если пиксель не белый
	for (int i = 0; i < infoBitmap.biHeight; i++)
	{
		for (int j = 0; j < infoBitmap.biWidth; j++)
		{
			if (arrayRGB[i][j].rgbBlue == 255 && arrayRGB[i][j].rgbRed == 255
				&& arrayRGB[i][j].rgbGreen == 255)
			{
				addArrayForRGB[i][j] = 1;
			}
			else addArrayForRGB[i][j] = 0;
		}
	}


	// Выполняем поиск области для гистограммы первой строки
	RECTANGLECOORDINATES m_Rectang = maxHistogramArea(addArrayForRGB[0],
infoBitmap.biWidth);

	int result = m_Rectang.area;
	for (int i = 1; i < infoBitmap.biHeight; i++)
	{
		// Создаем новую гистограмму для новой строки
		for (int j = 0; j < infoBitmap.biWidth; j++)
			if (addArrayForRGB[i][j])
				addArrayForRGB[i][j] += addArrayForRGB[i - 1][j];

		// Выполняем поиск области для текущей гистограммы
		RECTANGLECOORDINATES newRectang =
			maxHistogramArea(addArrayForRGB[i], infoBitmap.biWidth);

		// Сравниваем результаты и сохраняем структуру с максимальной областью
		if (newRectang.area >= result)
		{
			m_Rectang = newRectang;
			m_Rectang.y1 = i;
			result = m_Rectang.area;
		}

	}

	// Высчитываем координату Y0
	m_Rectang.y0 = m_Rectang.y1 - m_Rectang.y0 + 1;

	printf("%d\n%d\n%d\n%d\n", m_Rectang.x0, m_Rectang.y0, m_Rectang.x1, m_Rectang.y1);

	// Очистка памяти
	for (int i = 0; i < infoBitmap.biHeight; i++)
```

```
	{
		free(addArrayForRGB[i]);
	}
	free(addArrayForRGB);

}
```

2.5. Функция `maxHistogramArea`

Функция `maxHistogramArea`, находит максимальную прямоугольную площадь в заданной гистограмме и заполняет структуру `RECTANGLECOORDINATES`, полученными в результате обработки данными.

```
RECTANGLECOORDINATES maxHistogramArea(int* histogram, int length)
{
	RECTANGLECOORDINATES m_Rectang;
	int maxArea = 0;

	// Перебором находим максимальную площадь под гистограммой
	for (int i = 0; i < length; i++)
	{
		int lenghtYSize = histogram[i];
		for (int j = i; j < length; j++)
		{
			if (lenghtYSize > histogram[j])
			{
				lenghtYSize = histogram[j];
			}
			if ((j - i + 1) * lenghtYSize >= maxArea)
			{
				maxArea = (j - i + 1) * lenghtYSize;
				m_Rectang.x0 = i;
				m_Rectang.x1 = j;
				m_Rectang.y0 = lenghtYSize;
			}
		}
	}

	m_Rectang.area = maxArea;
	return m_Rectang;
}
```

2.6. Функция `isCorrectCommands`

Функция `isCorrectCommands` проверяет полученные команды из массива слов `data` на корректность.

Проверяется:

- Имя файла – один из 5 вариантов
- Координаты – они должны находится внутри области изображения.
- Команды – число от 1 до 4, причем 4 должно обязательно присутствовать.

В случае возникновения ошибки вызывается функция `writeError`, описание которое будет представлено ниже, и возвращается иной код возврата.

```
int isCorrectCommands(char** data, int lenght, BITMAPINFOHEADER infoBitmap)
{

	// Имя файла - один из 5 вариантов
	if (!strcmp(data[0], "./input_1.bmp") &&
		!strcmp(data[0], "./input_2.bmp") &&
		!strcmp(data[0], "./input_3.bmp") &&
		!strcmp(data[0], "./input_4.bmp") &&
		!strcmp(data[0], "./input_5.bmp"))
	{
		writeError(1);
		return 0;
	}

	int x0 = atoi(data[1]);
	int y0 = atoi(data[2]);
	int x1 = atoi(data[3]);
	int y1 = atoi(data[4]);

	// Если координаты области по X находятся вне изображения
	// или x0 < x1 - ошибка
	if (!(x0 < x1) || (x0 > infoBitmap.biWidth) || (x0 < 0))
	{
		writeError(2);
		return 0;
	}

	// Если координаты области по Y находятся вне изображения
	// или y0 > y1 - ошибка
	if (!(y0 > y1) || (y0 > infoBitmap.biHeight) || (y0 < 0))
	{
		writeError(3);
		return 0;
	}

	if ((x1 > infoBitmap.biWidth) || (x1 < 0))
	{
		writeError(4);
		return 0;
	}

	if ((y1 > infoBitmap.biHeight) || (y1 < 0))
	{
		writeError(5);
		return 0;
	}

	// Команда - число от 1 до 4 включительно
	// В противном случае - ошибка
	for (int i = 5; i < lenght; i++)
	{
		if ((atoi(data[i]) > 4) || (atoi(data[i]) < 1))
		{
			writeError(6);
			return 0;
		}
	}
	// 4 - последняя команда, в случае отсутствия - ошибка
	if (atoi(data[lenght - 1]) != 4)
	{
		writeError(6);
		return 0;
```

```
	}
}
```

2.7. Функция `writeError`

Функция `writeError` печатает соответствующую ошибку в зависимости от переданного кода ошибки:

1. Ошибка открытия изображения
2. Ошибка в координате x0
3. Ошибка в координате y0
4. Ошибка в координате x1
5. Ошибка в координате y1
6. Ошибка в командах

```
void writeError(int errorCode)
{
	printf("Fail with ");
	switch (errorCode)
	{
	case 1: printf("input_file"); break;
	case 2: printf("x0"); break;
	case 3: printf("y0"); break;
	case 4: printf("x1"); break;
	case 5: printf("y1"); break;
	case 6: printf("commands"); break;
	}
	printf("\n");
}
```

2.8. Функция `main`

В функции `main` считывается строка команд. Она разбивается на массив слов `data`, использую функцию `strtok`. После этого идет проверка данных на корректность, если все верно, тогда идет вызов функций в соответствии с номером команды, в противном случае – функция завершается с ненулевым кодом возврата.

```
int main()
{

	// Считывание строки команд
	char* string = (char*)malloc(sizeof(char) * 500);
	fgets(string, 500, stdin);
	(*strstr(string, "\n")) = 0;


	// Выделение памяти под массив лексем - команд
	char** data = (char**)malloc(sizeof(char*) * 25);
	for (int i = 0; i < 25; i++)
	{
		data[i] = NULL;
```

```
}

// Разделение строки на лексемы
int length = 0;
for (char* word = strtok(string, " "); word; word = strtok(NULL, " "))
{
	data[length++] = word;
}


// Открытие изображения BMP
FILE* inputFile = fopen(data[0], "rb");

// В случае отсутствия файла или ошибки открытия, вызов функции
// writeError и вывод ненулевого кода возврата
if (!inputFile)
{
	writeError(1);
	return 1;
}

// Инициализация структур BMP изображения и считывание их из файла
BITMAPFILEHEADER headerBitmap = readFromBitmapHeader(inputFile);
BITMAPINFOHEADER infoBitmap = readFromBitmapInfo(inputFile);

// Инициализация двумерного массива структур
RGBPIXEL** arrayRGB = (RGBPIXEL**)malloc(sizeof(RGBPIXEL*)*infoBitmap.biHeight);
for (int i = 0; i < infoBitmap.biHeight; i++)
{
	arrayRGB[i] = (RGBPIXEL*)malloc(sizeof(RGBPIXEL)*infoBitmap.biWidth);
}
arrayRGB = readFromBitmapRGB(inputFile, infoBitmap);

fclose(inputFile);

// Проверка переданных команд на корректность
if (!isCorrectCommands(data, length, infoBitmap))
{
	return 1;
}


// В соответствии с номером команды выполняется определенная функция
// 1 - поиск максимальной прямоугольной области
// 2 - нарисовать черный треугольник
// 3 - отображение изображения
// 4 - сохранение результатов обработки

for (int i = 5; i < length; i++)
{
	switch (atoi(data[i]))
	{
	case 1:
	{
		searchWhiteRectangle(arrayRGB, infoBitmap);
		break;
	}
	case 2:
	{
		drawBlackTriangle(arrayRGB, infoBitmap,
			atoi(data[1]), atoi(data[2]),
			atoi(data[3]), atoi(data[4]));
		break;
	}
```

```
		case 3:
		{
			mirrorReflectionBitmap(arrayRGB, infoBitmap);
			break;
		}
		case 4:
		{
			FILE* outFile = fopen("./refsol.bmp", "wb");

			// Запись данных BMP изображения
			writeToBitmapHeader(outFile, headerBitmap);
			writeToBitmapInfo(outFile, infoBitmap);
			writeToBitmapRGB(outFile, arrayRGB, infoBitmap);
			fclose(outFile);
			break;
		}
		}
	}

	// Освобождение памяти
	for (int i = 0; i < infoBitmap.biHeight; i++)
	{
		free(arrayRGB[i]);
	}

	free(arrayRGB);
	free(data);
	free(string);

	return 0;
}
```

2.9. `Makefile` для решения задачи

```
refsol: struct.h refsol.o rwBitmap.o
	gcc refsol.o -o refsol_main.out rwBitmap.o
	rm *.o
refsol.o: refsol.c struct.h
	gcc -c refsol.c
rwBitmap.o: rwBitmap.c rwBitmap.h struct.h
	gcc -c rwBitmap.c
```

3. Checker

При запуске скрипта первоначально запускается makefile для генератора и makefile для решения задачи. Полученные сгенерированные команды считываются построчно и передаются `refsol_main.out` и `usersol.c` (пример пользовательского решения данной задачи).

Результаты обработки записываются в соответствующие переменные, в случае несовпадения значений выводится ошибка, пользовательский вывод,

правильный вывод и команды. Если результаты совпадают, то проверяются изображения. Аналогично для изображений – если не совпадают, вывод ошибки.

```
#!/bin/bash

make -f makefile_generate
./generate_main.out
make -f makefile_refsol
gcc usersol.c

clear
count=1
cat input.txt | while read line
do
    refsol=`echo $line | ./refsol_main.out`
    if [[ -r refsol.bmp ]]
    then
           refsolbmp=`cat refsol.bmp`
    fi

    usersol=`echo $line | ./a.out`

    if [[ -r usersol.bmp ]]
    then
           userbmp=`cat usersol.bmp`
    fi

    if [[ "$refsol" != "$usersol" ]]
    then
           echo "Fail test N$count"
           echo "Input:"
           echo "$line"
           echo "Your output:"
           echo "$usersol"
           echo "Correct output:"
           echo "$refsol"
           exit 1
    else
           if [[ "$userbmp" != "$refsolbmp" ]]
           then
                  echo "Fail test N$count"
                  echo "Not correct BMP image"
                  exit 1
           fi
    fi
    echo "Test N$count - completed!"
    let "count+=1"
done


if [[ $? -ne 1 ]]
then
    echo "Successfully!"
else
    echo "Error!"
fi

rm -f *.txt *.out *.bmp
```

Пример использования программы

После запуска генератора, программа создает 5 изображений BMP и текстовый документ с командами. Пример сгенерированного изображения представлен на рисунке 2

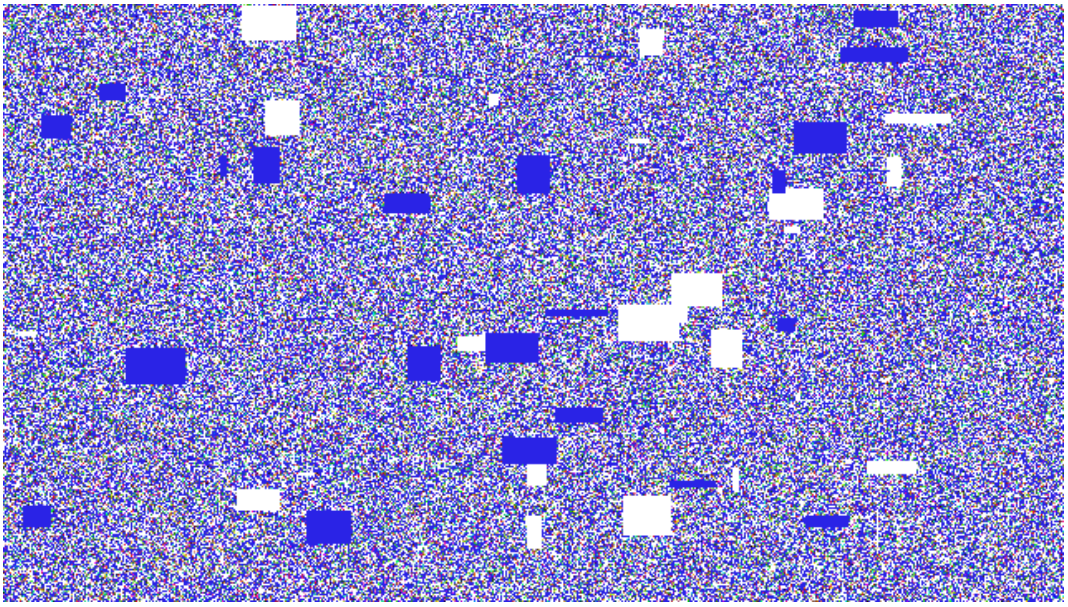

Рисунок 2. Файл input_1.bmp

После передачи команды *“./input_1.bmp 10 350 630 10 1 2 3 4”* `refsol.c` обрабатывается изображение, создается новое изображение под именем refsol.bmp (см. рис 3) и программа выводит “*144 1 176 21*”

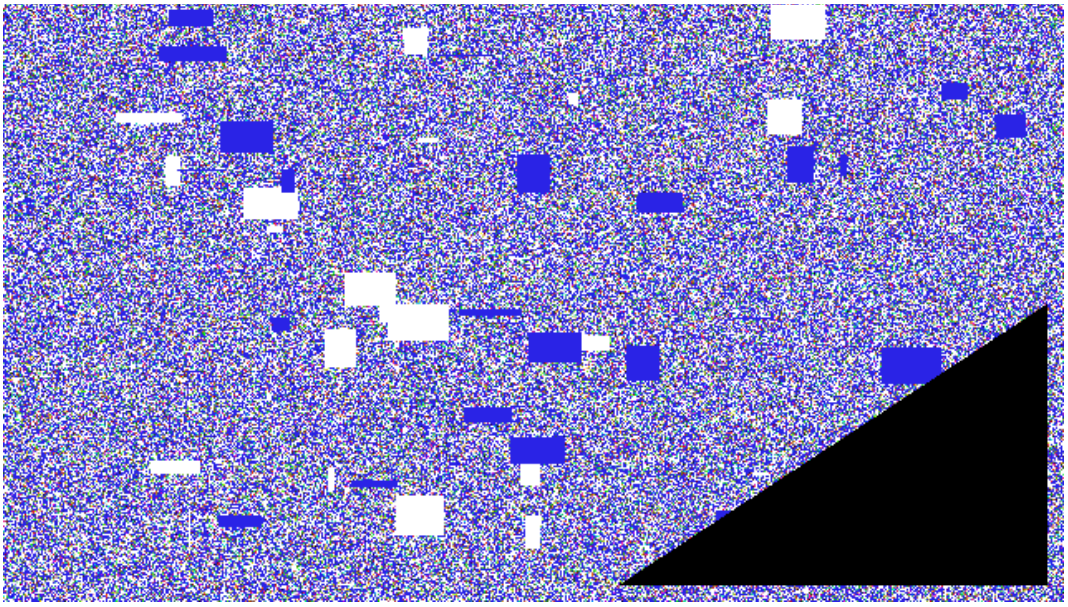

Рисунок 3. Файл refsol.bmp

Заключение

В ходе выполнения данной курсовой работы, были закреплены на практике знания о структурах и файлах, посредством создания функций для работы с ними на примере программы, обрабатывающей файл формата BMP, в котором используются специальные структуры данных.

Также были получены навыки работы с такими функциями как *atoi, strtok, srand*; была оптимизирована и улучшена работа с вводом и выводом файлов, работа со структурами, циклами и списками.

Получены навыки написания скриптов на *bash*, перенаправление вывода и вызов исполняемых файлов как последовательно, так и параллельно.

Было реализовано разбиение проекта на отдельные файлы, описывающей функций для работы с BMP файлом, что позволило удобнее добавлять и отлаживать функционал.

Список использованных источников

1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб: Издательство «Невский Диалект», 2001. 352 с.

2. UNIX. Программное окружение / Керниган Б., Пайк Р. СПб: Символ Плюс, 2003. 416с.

Приложение

Исходный код

Файл `struct.h`

```
#pragma pack(push)
#pragma pack(2)

/*=====================================================================
  Структура BITMAPFILEHEADER, содержащая
  информацию о типе, размере и макете файла
*/
typedef struct
{
	short 			bfType;		// Тип файла (должен быть BM)
	unsigned int 		bfSize;		// Размер файла
	unsigned short 		bfReserved1;	// Зарезервирован (должен быть 0)
	unsigned short 		bfReserved2;	// Зарезервирован (должен быть 0)
	unsigned int 		bfOffBits;	// Смещение от начала битов

} BITMAPFILEHEADER;
//=====================================================================



/*=====================================================================
  Структура BITMAPINFOHEADER, содержащая
  информацию о размерах и цветовом формате
*/
typedef struct
{
	unsigned int	biSize;		// Число байтов необходимое для структуры
	int		biWidth;	// Ширина изображения (в пикселях)
	int		biHeight;	// Высота изображения (в пикселях)
	unsigned short	biPlanes;	// Число плоскостей (должен быть 1)
	unsigned short	biBitCount;	// Число битов на пиксель (24 бита)
	unsigned int	biCompression;	// Тип сжатия
	unsigned int	biSizeImage;	// Размер изображения (в байтах)
	int		biXPelsPerMeter;// Разрешающая способность по горизонтали
	int		biYPelsPerMeter;// Разрешающая способность по вертикали
	unsigned int	biClrUsed;	// Количество используемых цветов
	unsigned int	biClrImportant;	// Количество существенных цветов

} BITMAPINFOHEADER;
//=====================================================================



/*=====================================================================
  Структура RGBPIXEL, содержащая информацию о пикселе изображения
*/
typedef struct
{
	unsigned char	rgbBlue;	// Синий   цвет (от 0 до 255)
	unsigned char	rgbGreen;	// Зеленый цвет (от 0 до 255)
	unsigned char	rgbRed;		// Красный цвет (от 0 до 255)

} RGBPIXEL;
//=====================================================================


#pragma pack(pop)
```

Файл rwBitmap.h

```
/*=====================================================================
  Функция WriteToBitmapHeader записывает в файл outputFile
  структуру данных headerBitmap
*/
void writeToBitmapHeader(FILE* outfile, BITMAPFILEHEADER headerBMP);
//=====================================================================


/*=====================================================================
  Функция WriteToBitmapInfo записывает в файл outputFile
  структуру данных infoBitmap
*/
void writeToBitmapInfo(FILE* outfile, BITMAPINFOHEADER infoBMP);
//=====================================================================


/*=====================================================================
  Функция WriteToBitmapRGB записывает в файл outputFile матрицу пикселей
  arrayRGB, основываясь на размеры изображения из структуры infoBitmap
*/
void writeToBitmapRGB(FILE* outfile, RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBMP);
//=====================================================================


/*=====================================================================
  Функция ReadFromBitmapHeader считывает из файла inputFile
  структуру headerBitmap и возвращает ее
*/
BITMAPFILEHEADER readFromBitmapHeader(FILE* bitmap);
//=====================================================================



/*=====================================================================
  Функция ReadFromBitmapInfo считывает из файла inputFile
  структуру infoBitmap и возвращает ее
*/
BITMAPINFOHEADER readFromBitmapInfo(FILE* bitmap);
//=====================================================================



/*=====================================================================
  Функция ReadFromBitmapRGB считывает из файла матрицу
  пикселей с учетом размеров изображения и возвращает ее
*/
RGBPIXEL** readFromBitmapRGB(FILE* bitmap, BITMAPINFOHEADER info);
//=====================================================================
```

Файл rwBitmap.c

```
#include <stdio.h>
#include <stdlib.h>
#include "struct.h"

void writeToBitmapHeader(FILE* outputFile, BITMAPFILEHEADER headerBitmap)
{
	fwrite(&headerBitmap, sizeof(BITMAPFILEHEADER), 1, outputFile);
}
```

```
void writeToBitmapInfo(FILE* outputFile, BITMAPINFOHEADER infoBitmap)
{
	fwrite(&infoBitmap, sizeof(BITMAPINFOHEADER), 1, outputFile);
}

void writeToBitmapRGB(FILE* outputFile, RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap)
{

	int extraBytes = 4 - ((infoBitmap.biWidth * 3) % 4);
	if (extraBytes == 4)
	{
		extraBytes = 0;
	}

	for (int i = infoBitmap.biHeight - 1; i >= 0; i--)
	{
		for (int j = 0; j < infoBitmap.biWidth; j++)
		{
			fwrite(&arrayRGB[i][j].rgbBlue, 1, 1, outputFile);
			fwrite(&arrayRGB[i][j].rgbGreen, 1, 1, outputFile);
			fwrite(&arrayRGB[i][j].rgbRed, 1, 1, outputFile);
		}

		if (extraBytes)
		{
			for (int j = 0; j < extraBytes; j++)
			{
				fwrite("0", 1, 1, outputFile);
			}
		}
	}

}

BITMAPFILEHEADER readFromBitmapHeader(FILE* inputFile)
{
	BITMAPFILEHEADER headerBitmap;
	fread(&headerBitmap, sizeof(BITMAPFILEHEADER), 1, inputFile);
	return headerBitmap;
}

BITMAPINFOHEADER readFromBitmapInfo(FILE* inputFile)
{
	BITMAPINFOHEADER infoBitmap;
	fread(&infoBitmap, sizeof(BITMAPINFOHEADER), 1, inputFile);
	return infoBitmap;
}

RGBPIXEL** readFromBitmapRGB(FILE* inputFile, BITMAPINFOHEADER infoBitmap)
{
	RGBPIXEL** arrayRGB = (RGBPIXEL**)malloc(sizeof(RGBPIXEL*)*infoBitmap.biHeight);
	for (int i = 0; i < infoBitmap.biHeight; i++)
	{
		arrayRGB[i] = (RGBPIXEL*)malloc(sizeof(RGBPIXEL)*infoBitmap.biWidth);
	}

	int extraBytes = 4 - ((infoBitmap.biWidth * 3) % 4);
	if (extraBytes == 4)
	{
		extraBytes = 0;
	}

	for (int i = infoBitmap.biHeight - 1; i >= 0; i--)
	{
```

```
            for (int j = 0; j < infoBitmap.biWidth; j++)
            {
                  fread(&arrayRGB[i][j].rgbBlue, 1, 1, inputFile);
                  fread(&arrayRGB[i][j].rgbGreen, 1, 1, inputFile);
                  fread(&arrayRGB[i][j].rgbRed, 1, 1, inputFile);
            }

            if (extraBytes)
            {
                  for (int j = 0; j < extraBytes; j++)
                  {
                        char zeroByte = '0';
                        fread(&zeroByte, 1, 1, inputFile);
                  }
            }
      }

      return arrayRGB;

}
```

Файл generate.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "struct.h"
#include "rwBitmap.h"


#define WIDTH      640         // Ширина изображения
#define HEIGHT     360         // Высота изображения
#define TYPE       0x4D42      // Тип файла (должен быть BM)
#define SIZEINFO   40          // Размер структуры BITMAPINFOHEADER
#define SIZEHEADER 14          // Размер структуры BITMAPFILEHEADER
#define BITCOUNT   24          // Число битов на пиксель (24 бита)


/*=======================================================================

  Функция addColorToBitmap заполняет переданный массив структур,
  отвечающий за пиксели изображения, произвольным образом данными
  одним из трех вариантов: белым цветом, переданным цветом фона
  или абсолютно произвольным RGB цветом
*/
void addColorToBitmap(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBMP,
      int desiredColorRed, int desiredColorGreen, int desiredColorBlue);
//=======================================================================



/*=======================================================================

  Функция addRectangleToBitmap заполняет переданный массив структур,
  отвечающий за пиксели изображения, произвольными прямоугольниками.
*/
void addRectangleToBitmap(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBMP,
      int numberOfRectangles, int desiredColorRed, int desiredColorGreen,
      int desiredColorBlue);
//=======================================================================
```

```
/*=========================================================================

  Функция createBitmapFile создает BMP изображение размерами
  WIDTH * HEIGHT, с номером fileNameIndex и цветом фона desiredColorRed,
  desiredColorGreen, desiredColorBlue, переданными данной функции
*/
void createBitmapFile(char* fileNameIndex, int desiredColorRed,
	int desiredColorGreen, int desiredColorBlue);
//=========================================================================



/*=========================================================================
  Функция createTxtFile создает файл input.txt с командами.
  Количество команд - 50.
*/
void createTxtFile();
//=========================================================================

int main()
{
	srand(time(NULL));

	unsigned int desiredColorRed, desiredColorGreen, desiredColorBlue;

	char fileNameIndex[3];  // Номер изображения

	for (int i = 1; i <= 5; i++)
	{
		// Преобразование номера изображения в строку
		sprintf(fileNameIndex, "%d", i);

		desiredColorRed = rand() % 255;     //
		desiredColorGreen = rand() % 255;   // Генерация цвета фона
		desiredColorBlue = rand() % 255;    //

		// Создание изображения
		createBitmapFile(fileNameIndex, desiredColorRed, desiredColorGreen,
			desiredColorBlue);
	}

	// Создание файла с командами
	createTxtFile();

	return 0;
}


void createBitmapFile(char* fileNameIndex, int desiredColorRed,
	int desiredColorGreen, int desiredColorBlue)
{
	char fileName[30] = "./input_";   // Создание имени изображения
	strcat(fileName, fileNameIndex);  // с учетом его номера
	strcat(fileName, ".bmp");         //

	FILE* outfileBMP = fopen(fileName, "wb");

	BITMAPFILEHEADER headerBMP;
	BITMAPINFOHEADER infoBMP;

	// Количество байтов необходимых для выравнивания изображения
```

```
	int extraBytes = 4 - ((WIDTH * 3) % 4);
	if (extraBytes == 4)
	{
		extraBytes = 0;
	}

	// Полный размер изображения с учетом байтов для выравнивания
	int paddedSize = ((WIDTH * 3) + extraBytes) * HEIGHT;

	// Заполнение структур BMP изображения
	headerBMP.bfType = TYPE;
	headerBMP.bfSize = SIZEINFO + SIZEHEADER + paddedSize;
	headerBMP.bfReserved1 = 0;
	headerBMP.bfReserved2 = 0;
	headerBMP.bfOffBits = SIZEINFO + SIZEHEADER;

	infoBMP.biSize = SIZEINFO;
	infoBMP.biWidth = WIDTH;
	infoBMP.biHeight = HEIGHT;
	infoBMP.biPlanes = 1;
	infoBMP.biBitCount = BITCOUNT;
	infoBMP.biCompression = 0;
	infoBMP.biSizeImage = paddedSize;
	infoBMP.biXPelsPerMeter = 0;
	infoBMP.biYPelsPerMeter = 0;
	infoBMP.biClrUsed = 0;
	infoBMP.biClrImportant = 0;

	// Инициализация двумерного массива структур
	RGBPIXEL** arrayRGB = (RGBPIXEL**)malloc(sizeof(RGBPIXEL*)*HEIGHT);
	for (int i = 0; i < HEIGHT; i++)
	{
		arrayRGB[i] = (RGBPIXEL*)malloc(sizeof(RGBPIXEL)*WIDTH);
	}

	// Вызов функции addColorToBitmap
	addColorToBitmap(arrayRGB, infoBMP, desiredColorRed,
		desiredColorGreen, desiredColorBlue);

	// Вызов функций addRectangleToBitmap
	addRectangleToBitmap(arrayRGB, infoBMP, 15, 255, 255, 255);
	addRectangleToBitmap(arrayRGB, infoBMP, 15, desiredColorRed,
		desiredColorGreen, desiredColorBlue);

	// Запись данных BMP изображения
	writeToBitmapHeader(outfileBMP, headerBMP);
	writeToBitmapInfo(outfileBMP, infoBMP);
	writeToBitmapRGB(outfileBMP, arrayRGB, infoBMP);

	// Освобождение памяти
	for (int i = 0; i < HEIGHT; i++)
	{
		free(arrayRGB[i]);
	}

	free(arrayRGB);

	fclose(outfileBMP);
}

void addColorToBitmap(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBMP,
	int desiredColorRed, int desiredColorGreen, int desiredColorBlue)
{
```

```
	for (int i = 0; i < infoBMP.biHeight; i++)
	{
		for (int j = 0; j < infoBMP.biWidth; j++)
		{

			// Переменная, отвечающая за цвет пикселя
			// 0 - белый, 1 - рандомный, 2 - переданный
			int colorFrequency;
			colorFrequency = rand() % 3;

			if (colorFrequency == 0)
			{
				arrayRGB[i][j].rgbBlue = 255;	// Заполнение
				arrayRGB[i][j].rgbGreen = 255;	// пикселя белым
				arrayRGB[i][j].rgbRed = 255;	// цветом
			}

			else if (colorFrequency == 1)
			{
				arrayRGB[i][j].rgbBlue = rand() % 256;	// Заполнение
				arrayRGB[i][j].rgbGreen = rand() % 256;	// пикселя рандомным
				arrayRGB[i][j].rgbRed = rand() % 256;	// цветом
			}

			else
			{
				arrayRGB[i][j].rgbBlue = desiredColorBlue;	// Заполнение
				arrayRGB[i][j].rgbGreen = desiredColorGreen;	// пикселя
переданным
				arrayRGB[i][j].rgbRed = desiredColorRed;	// цветом
			}

		}
	}

}

void addRectangleToBitmap(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBMP,
	int numberOfRectangles, int desiredColorRed, int desiredColorGreen,
	int desiredColorBlue)
{

	for (int k = 0; k < numberOfRectangles; k++)
	{
		// Инициализация размеров добавляемых прямоуголников
		int RectangleX0Coordinates = rand() % (infoBMP.biWidth * 13 / 15 - 1);
		int RectangleY0Coordinates = rand() % (infoBMP.biHeight * 13 / 15 - 1);
		int RectangleWidth = rand() % (infoBMP.biWidth / 10) + 1;
		int RectangleHeight = rand() % (infoBMP.biHeight / 10) + 1;

		for (int i = RectangleY0Coordinates;
			i < RectangleY0Coordinates + RectangleHeight; i++)
		{
			for (int j = RectangleX0Coordinates;
				j < RectangleX0Coordinates + RectangleWidth; j++)
			{
				arrayRGB[i][j].rgbBlue = desiredColorBlue;	// Заполнение
				arrayRGB[i][j].rgbGreen = desiredColorGreen;	// пикселя
переданным
				arrayRGB[i][j].rgbRed = desiredColorRed;	// цветом
			}
		}
	}
```

```
}

void createTxtFile()
{
	FILE* outfileTXT = fopen("./input.txt", "w");

	// Запись правильных команд первого теста
	fprintf(outfileTXT, "./input_1.bmp 10 50 20 30 1 2 3 3 2 1 4\n");

	for (int i = 1; i < 50; i++)
	{
		// Переменная, отвечающая за количество команд
		int commandsSize = rand() % 12 + 1;


		// Запись пути BMP изображения
		fprintf(outfileTXT, "%s", "./input_");

		// Переменная isCorrectCommand отвечает за
		// корректность генерируемых команд
		// 0 - некорректные, 1 - корректные
		int isCorrectCommand = rand() % 2;


		if (isCorrectCommand)
		{
			// Запись правильного индекса изображения
			fprintf(outfileTXT, "%d.bmp ", rand() % 5 + 1);
		}
		else
		{
			// Запись неправильного индекса изображения
			fprintf(outfileTXT, "%d.bmp ", rand() % 10 + 1);
		}


		isCorrectCommand = rand() % 2;
		if (!isCorrectCommand)
		{
			// Запись неправильных координат области изображения
			fprintf(outfileTXT, "%d %d ", rand() % 5000 - 1000, rand() % 5000 -
1000);
			fprintf(outfileTXT, "%d %d ", rand() % 5000 - 1000, rand() % 5000 -
1000);
		}
		else
		{
			// Генерация корректных команд

			int x0 = 1;
			int y0 = 0;
			int x1 = 0;
			int y1 = 1;

			// Генерация коррекнтых координат по X
			// X0 должен быть больше, чем X1
			while (x0 >= x1)
			{
				x0 = rand() % WIDTH;
				x1 = rand() % HEIGHT;
			}

			// Генерация коррекнтых координат по Y
			// Y1 должен быть больше, чем Y0
```

```
                while (y1 >= y0)
                {
                        y0 = rand() % WIDTH;
                        y1 = rand() % HEIGHT;
                }

                // Запись правильных координат области изображения
                fprintf(outfileTXT, "%d %d ", x0, y0);
                fprintf(outfileTXT, "%d %d ", x1, y1);
        }


        isCorrectCommand = rand() % 2;
        if (!isCorrectCommand)
        {
                // Запись неправильных команд
                for (int i = 0; i < commandsSize; i++)
                {
                        fprintf(outfileTXT, "%d ", rand() % 7);
                }

                fprintf(outfileTXT, "\n");
        }
        else
        {
                // Запись правильных команд
                for (int i = 0; i < commandsSize; i++)
                {
                        fprintf(outfileTXT, "%d ", rand() % 3 + 1);
                }

                fprintf(outfileTXT, "%d\n", 4);
        }
    }

    fclose(outfileTXT);
}
```

Файл `refsol.c`

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "struct.h"
#include "rwBitmap.h"


/*=====================================================================
Функция mirrorReflectionBitmap отражает по горизонтали двумерный
массив цветов arrayRGB с учетом размеров изображения BMP из
структуры infoBitmap
*/
void mirrorReflectionBitmap(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap);
//=====================================================================


/*=====================================================================
Функция drawBlackTriangle рисует прямоугольный треугольник в массиве
arrayRGB в заданной области (x0, y0, x1, y1) черным цветом. Прямой
угол лежит в левом нижнем углу области, длина катетов определяется
длиной стороны области, к которой они прилегают, и равна половине
```

```
этой стороны.
*/
void drawBlackTriangle(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap,
	int x0, int y0, int x1, int y1);
//=========================================================================


/*=========================================================================
Функция searchWhiteRectangle находит в массиве arrayRGB самый
большой белый прямоугольник и выводит координаты левого верхнего
угла и правого нижнего.
*/
void searchWhiteRectangle(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap);
//=========================================================================


/*=========================================================================
Функция writeError печатает "Fail with ... " в зависимости от
переданного аргумента errorCode (1 - input_file, 2 - x0, 3 - y0,
4 - x1, 5 - y1, 6 - commands)
*/
void writeError(int errorCode);
//=========================================================================


/*=========================================================================
Функция isCorrectCommands проверяет корректность введенных данных
массива слов (команд) data длиной lenght. В случае ошибки вызывает
функцию writeError и возвращает 1, в противном случае - 0
*/
int isCorrectCommands(char** data, int lenght, BITMAPINFOHEADER infoBitmap);
//=========================================================================


/*=========================================================================
Структура RECTANGLECOORDINATES описывает параметры найденной
прямоугольной белой области: координаты по оси X,Y и площадь
*/
typedef struct
{
	int x0;
	int y0;
	int x1;
	int y1;
	int area;
} RECTANGLECOORDINATES;
//=========================================================================


/*=========================================================================
Функция maxHistogramArea ищет максимальную прямоугольную площадь
под заданной гистограммой histogram и возвращает структуру
RECTANGLECOORDINATES, заполненную координатами по оси X и
длину прямоугольника по оси Y
*/
RECTANGLECOORDINATES maxHistogramArea(int* histogram, int Height);
//=========================================================================


int main()
{

	// Считывание строки команд
	char* string = (char*)malloc(sizeof(char) * 500);
```

```
fgets(string, 500, stdin);
(*strstr(string, "\n")) = 0;


// Выделение памяти под массив лексем - команд
char** data = (char**)malloc(sizeof(char*) * 25);
for (int i = 0; i < 25; i++)
{
	data[i] = NULL;
}

// Разделение строки на лексемы
int length = 0;
for (char* word = strtok(string, " "); word; word = strtok(NULL, " "))
{
	data[length++] = word;
}


// Открытие изображения BMP
FILE* inputFile = fopen(data[0], "rb");

// В случае отсутствия файла или ошибки открытия, вызов функции
// writeError и вывод ненулевого кода возврата
if (!inputFile)
{
	writeError(1);
	return 1;
}

// Инициализация структур BMP изображения и считывание их из файла
BITMAPFILEHEADER headerBitmap = readFromBitmapHeader(inputFile);
BITMAPINFOHEADER infoBitmap = readFromBitmapInfo(inputFile);

// Инициализация двумерного массива структур
RGBPIXEL** arrayRGB = (RGBPIXEL**)malloc(sizeof(RGBPIXEL*)*infoBitmap.biHeight);
for (int i = 0; i < infoBitmap.biHeight; i++)
{
	arrayRGB[i] = (RGBPIXEL*)malloc(sizeof(RGBPIXEL)*infoBitmap.biWidth);
}
arrayRGB = readFromBitmapRGB(inputFile, infoBitmap);

fclose(inputFile);

// Проверка переданных команд на корректность
if (!isCorrectCommands(data, length, infoBitmap))
{
	return 1;
}


// В соответствии с номером команды выполняется определенная функция
// 1 - поиск максимальной прямоугольной области
// 2 - нарисовать черный треугольник
// 3 - отображение изображения
// 4 - сохранение результатов обработки

for (int i = 5; i < length; i++)
{
	switch (atoi(data[i]))
	{
	case 1:
	{
		searchWhiteRectangle(arrayRGB, infoBitmap);
```

```
                        break;
                }
                case 2:
                {
                        drawBlackTriangle(arrayRGB, infoBitmap,
                                atoi(data[1]), atoi(data[2]),
                                atoi(data[3]), atoi(data[4]));
                        break;
                }
                case 3:
                {
                        mirrorReflectionBitmap(arrayRGB, infoBitmap);
                        break;
                }
                case 4:
                {
                        FILE* outFile = fopen("./refsol.bmp", "wb");

                        // Запись данных BMP изображения
                        writeToBitmapHeader(outFile, headerBitmap);
                        writeToBitmapInfo(outFile, infoBitmap);
                        writeToBitmapRGB(outFile, arrayRGB, infoBitmap);
                        fclose(outFile);
                        break;
                }
                }
        }

        // Освобождение памяти
        for (int i = 0; i < infoBitmap.biHeight; i++)
        {
                free(arrayRGB[i]);
        }

        free(arrayRGB);
        free(data);
        free(string);

        return 0;
}


void mirrorReflectionBitmap(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap)
{
        RGBPIXEL temp;  // Переменная для обмена пикселями

        for (int i = 0; i < infoBitmap.biHeight; i++)
        {
                for (int j = 0; j < infoBitmap.biWidth / 2; j++)
                {
                        // Обмениваем симметричные элементы местами
                        temp = arrayRGB[i][j];
                        arrayRGB[i][j] = arrayRGB[i][infoBitmap.biWidth - 1 - j];
                        arrayRGB[i][infoBitmap.biWidth - 1 - j] = temp;
                }
        }

}

void drawBlackTriangle(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap,
        int x0, int y0, int x1, int y1)
{

        // Массив steps содержит длины отрезков i - строки треугольника
```

```
	// От 0 до половины стороны катета, прилежащего к оси X
	float* steps = (float*)malloc(sizeof(float)*(y0 - y1));
	steps[0] = 0.0;

	// Расчет длин отрезков
	for (int i = 1; i < y0 - y1; i++)
	{
		steps[i] = (float)(x1 - x0) / (float)(y0 - y1) + steps[i - 1];
	}

	int k = 0;
	for (int i = (y1 + y0) / 2; i < y0; i++)
	{
		for (int j = x0; j < x0 + (int)steps[k]; j++)
		{
			arrayRGB[i][j].rgbBlue = 0;	// Заполнение
			arrayRGB[i][j].rgbRed = 0;	// пикселя черным
			arrayRGB[i][j].rgbGreen = 0;	// цветом
		}
		k++;
	}

}

void searchWhiteRectangle(RGBPIXEL** arrayRGB, BITMAPINFOHEADER infoBitmap)
{

	// Создаем вспомогательный двумерный массив с такими же размерами, что и изображение
	int** addArrayForRGB = (int**)malloc(sizeof(int**)*infoBitmap.biHeight);
	for (int i = 0; i < infoBitmap.biHeight; i++)
	{
		addArrayForRGB[i] = (int*)malloc(sizeof(int)*infoBitmap.biWidth);
	}


	// Заполняем его следующим образом: 1 - если пиксель белый
	//                                                    0 - если пиксель не белый
	for (int i = 0; i < infoBitmap.biHeight; i++)
	{
		for (int j = 0; j < infoBitmap.biWidth; j++)
		{
			if (arrayRGB[i][j].rgbBlue == 255 && arrayRGB[i][j].rgbRed == 255
				&& arrayRGB[i][j].rgbGreen == 255)
			{
				addArrayForRGB[i][j] = 1;
			}
			else addArrayForRGB[i][j] = 0;
		}
	}
	// Выполняем поиск области для гистограммы первой строки
	RECTANGLECOORDINATES m_Rectang = maxHistogramArea(addArrayForRGB[0],
infoBitmap.biWidth);

	int result = m_Rectang.area;
	for (int i = 1; i < infoBitmap.biHeight; i++)
	{
		// Создаем новую гистограмму для новой строки
		for (int j = 0; j < infoBitmap.biWidth; j++)
			if (addArrayForRGB[i][j])
				addArrayForRGB[i][j] += addArrayForRGB[i - 1][j];

		// Выполняем поиск области для текущей гистограммы
		RECTANGLECOORDINATES newRectang =
			maxHistogramArea(addArrayForRGB[i], infoBitmap.biWidth);
```

```
			// Сравниваем результаты и сохраняем структуру с максимальной областью
			if (newRectang.area >= result)
			{
				m_Rectang = newRectang;
				m_Rectang.y1 = i;
				result = m_Rectang.area;
			}

		}

		// Высчитываем координату Y0
		m_Rectang.y0 = m_Rectang.y1 - m_Rectang.y0 + 1;

		printf("%d\n%d\n%d\n%d\n", m_Rectang.x0, m_Rectang.y0, m_Rectang.x1, m_Rectang.y1);

		// Очистка памяти
		for (int i = 0; i < infoBitmap.biHeight; i++)
		{
			free(addArrayForRGB[i]);
		}
		free(addArrayForRGB);

}

RECTANGLECOORDINATES maxHistogramArea(int* histogram, int length)
{
		RECTANGLECOORDINATES m_Rectang;
		int maxArea = 0;

		// Перебором находим максимальную площадь под гистограммой
		for (int i = 0; i < length; i++)
		{
			int lenghtYSize = histogram[i];
			for (int j = i; j < length; j++)
			{
				if (lenghtYSize > histogram[j])
				{
					lenghtYSize = histogram[j];
				}
				if ((j - i + 1) * lenghtYSize >= maxArea)
				{
					maxArea = (j - i + 1) * lenghtYSize;
					m_Rectang.x0 = i;
					m_Rectang.x1 = j;
					m_Rectang.y0 = lenghtYSize;
				}
			}
		}

		m_Rectang.area = maxArea;
		return m_Rectang;
}

void writeError(int errorCode)
{
		printf("Fail with ");
		switch (errorCode)
		{
		case 1: printf("input_file"); break;
		case 2: printf("x0"); break;
		case 3: printf("y0"); break;
		case 4: printf("x1"); break;
		case 5: printf("y1"); break;
```

```
        case 6: printf("commands"); break;
        }
        printf("\n");
}

int isCorrectCommands(char** data, int lenght, BITMAPINFOHEADER infoBitmap)
{

        // Имя файла - один из 5 вариантов
        if (!strcmp(data[0], "./input_1.bmp") &&
                !strcmp(data[0], "./input_2.bmp") &&
                !strcmp(data[0], "./input_3.bmp") &&
                !strcmp(data[0], "./input_4.bmp") &&
                !strcmp(data[0], "./input_5.bmp"))
        {
                writeError(1);
                return 0;
        }

        int x0 = atoi(data[1]);
        int y0 = atoi(data[2]);
        int x1 = atoi(data[3]);
        int y1 = atoi(data[4]);

        // Если координаты области по X находятся вне изображения
        // или x0 < x1 - ошибка
        if (!(x0 < x1) || (x0 > infoBitmap.biWidth) || (x0 < 0))
        {
                writeError(2);
                return 0;
        }

        // Если координаты области по Y находятся вне изображения
        // или y0 > y1 - ошибка
        if (!(y0 > y1) || (y0 > infoBitmap.biHeight) || (y0 < 0))
        {
                writeError(3);
                return 0;
        }

        if ((x1 > infoBitmap.biWidth) || (x1 < 0))
        {
                writeError(4);
                return 0;
        }

        if ((y1 > infoBitmap.biHeight) || (y1 < 0))
        {
                writeError(5);
                return 0;
        }

        // Команда - число от 1 до 4 включительно
        // В противном случае - ошибка
        for (int i = 5; i < lenght; i++)
        {
                if ((atoi(data[i]) > 4) || (atoi(data[i]) < 1))
                {
                        writeError(6);
                        return 0;
                }
        }

        // 4 - последняя команда, в случае отсутствия - ошибка
```

```
	if (atoi(data[lenght - 1]) != 4)
	{
		writeError(6);
		return 0;
	}

}
```

Файл makefile_generate

```
generate: struct.h generate.o rwBitmap.o
	gcc generate.o -o  generate_main.out rwBitmap.o
	rm *.o
generate.o: generate.c struct.h
	gcc -c generate.c
rwBitmap.o: rwBitmap.c rwBitmap.h struct.h
	gcc -c rwBitmap.c
```

Файл makefile_refsol

```
refsol: struct.h refsol.o rwBitmap.o
	gcc refsol.o -o refsol_main.out rwBitmap.o
	rm *.o
refsol.o: refsol.c struct.h
	gcc -c refsol.c
rwBitmap.o: rwBitmap.c rwBitmap.h struct.h
	gcc -c rwBitmap.c
```

Файл script.sh

```
#!/bin/bash

make -f makefile_generate
./generate_main.out
make -f makefile_refsol
gcc usersol.c

clear
count=1
cat input.txt | while read line
do
    refsol=`echo $line | ./refsol_main.out`
    if [[ -r refsol.bmp ]]
    then
        refsolbmp=`cat refsol.bmp`
    fi

    usersol=`echo $line | ./a.out`

    if [[ -r usersol.bmp ]]
    then
        userbmp=`cat usersol.bmp`
    fi

    if [[ "$refsol" != "$usersol" ]]
    then
        echo "Fail test N$count"
        echo "Input:"
        echo "$line"
```

```
            echo "Your output:"
            echo "$usersol"
            echo "Correct output:"
            echo "$refsol"
            exit 1
    else
            if [[ "$userbmp" != "$refsolbmp" ]]
            then
                    echo "Fail test N$count"
                    echo "Not correct BMP image"
                    exit 1
            fi
    fi
    echo "Test N$count - completed!"
    let "count+=1"
done

if [[ $? -ne 1 ]]
then
    echo "Successfully!"
else
    echo "Error!"
fi

rm -f *.txt *.out *.bmp
```