**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра MO ЭВМ**

Курсовая РАБОТА

**по дисциплине «ПРОГРАММИРОВАНИЕ»**

Тема: BMP - файлы

|  |  |  |
| --- | --- | --- |
| Студент гр. 6303 |  | Жахин А. А. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Жахин А. А. | | |
| Группа 6303 | | |
| Тема работы: BMP-файл | | |
|  | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание структур BMP-файла * Проверка на корректность введенных данных * Реализация функций программы * Примеры работы программы * Заключение * Список использованных источников * Приложение А. Исходный код программы | | |
| Предполагаемый объем пояснительной записки:  Не менее 10 страниц. | | |
| Дата выдачи задания: 28.04.2017 | | |
| Дата сдачи реферата: 2.06.2017 | | |
| Дата защиты реферата: 2.06.2017 | | |
| Студент |  | Жахин А. А. |
| Преподаватель |  | Берленко Т.А. |

**Аннотация**

В результате курсовой работы была реализована программа на языке программирования С, которая делит заданную область BMP-файла на 4 равные части и по диагонали меняет их друг с другом местами, а также были изучены методы работы с BMP-файлами, разработан алгоритм построения необходимого изображения.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Описание структур BMP-файла | 6 |
| 1.1. | Общее строение | 6 |
| 1.2.  1.3  1.4  1.5  1.6 | BITMAPFILEHEADER  BITMAPINFOHEADER  RGBTRIPLE  Pragma  Типы данных | 6  6  7  7  8 |
| 2. | Ход работы | 9 |
| 2.1. | Проверка координат на корректность | 9 |
| 2.2.  2.3  2.4 | Проверка существования файла  Функция swap  Функция main | 9  10  12 |
| 3. | Примеры работы программы | 14 |
| 4. | Заключение | 16 |
|  | Список использованных источников | 17 |
|  | Приложение А. Исходный код | 18 |
|  |  |  |

**введение**

* Требуется написать программу, которая делит заданную область BMP-файла на 4 равные части и меняет их местами по диагонали. Результат сохраняет в новом файле.
* Программа получает параметры из входного потока и должна проверить их корректность. Параметры:
* input\_file
* x0
* y0
* x1
* y1
* input\_file - имя BMP файла
* x0 y0 левый верхний угол области (отсчет с точки 0, 0)
* x1 y1 правый нижний угол области
* В случае, если программа получила некорректные параметры, то:
* не создается выходного в файла
* выводится сообщение об ошибке “Fail with <имя параметра>”.

### Общие сведения

* 24 бита на цвет
* без сжатия
* файл всегда соответствует формату (проверять не нужно)
* обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
* обратите внимание на порядок записи пикселей
* все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).**1. ОПИСАНИЕ СТРУКТУР BMP-файла**

**1.1. Общее строение BMP-файла**

**BMP** (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Bitmap Picture*) — формат хранения растровых изображений, разработанный компанией [Microsoft](https://ru.wikipedia.org/wiki/Microsoft).

Для начала приведем графическое представление данных в bmp (рис. 1)

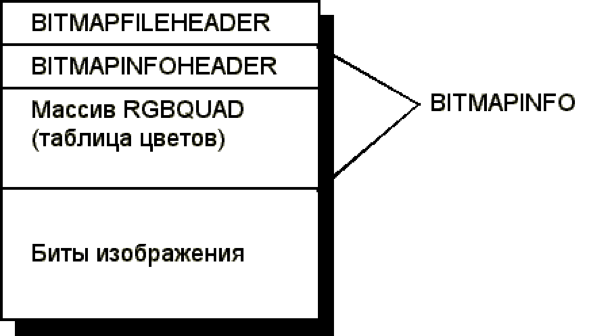


Рис. 1

**1.2. BITMAPFILEHEADER**

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

|  |
| --- |
| typedef struct BITMAPFILEHEADER  {  WORD bfType;  DWORD bfSize;  WORD bfReserved1;  WORD bfReserved2;  DWORD bfOffBits;  } BITMAPFILEHEADER; |

bfType определяет тип файла. Здесь он должен быть BM.   
bfSize - это размер самого файла в байтах.

bfReserved1 и bfReserved2 зарезервированы и должны быть нулями.

bfOffBits. Это один из самых важных полей в этой структуре. Он показывает, где начинается сам битовый массив относительно начала файла (или, как написано в MSDN, "от начала структуры BITMAPFILEHEADER")

**1.3. BITMAPINFOHEADER**

Дальше идет структура BITMAPINFOHEADER, которая объявлена так:

|  |
| --- |
| typedef struct BITMAPINFOHEADER  {  DWORD biSize;  LONG biWidth;  LONG biHeight;  WORD biPlanes;  WORD biBitCount;  DWORD biCompression;  DWORD biSizeImage;  LONG biXPelsPerMeter;  LONG biYPelsPerMeter;  DWORD biClrUsed;  DWORD biClrImportant;  } BITMAPINFOHEADER; |

biSize - это размер самой структуры. Ее нужно инициализировать следующим образом: bih.biSize = sizeof (BITMAPINFOHEADER);

biWidth и biHeight задают соответственно ширину и высоту картинки в пикселях.   
biPlanes задает количество плоскостей. Пока оно всегда устанавливается в 1.

biBitCount - Количество бит на один пиксель.

biCompression обозначает тип сжатия.

biSizeImage обозначает размер картинки в байтах.

biXPelsPerMeter и biYPelsPerMeter обозначают соответственно горизонтальное и вертикальное разрешение (в пикселях на метр) конечного устройства, на которое будет выводиться битовый массив (растр)

biClrUsed определяет количество используемых цветов из таблицы. Если это значение равно нулю, то в растре используется максимально возможное количество цветов, которые разрешены значением biBitCount.

biClrImportant - это количество важных цветов.

**1.4. RGBTRIPLE**

*biBitCount* = 24 это самый простой формат. Здесь 3 байта определяют 3 компоненты цвета. То есть по компоненте на байт. Просто читаем по структуре RGBTRIPLE и используем его поля *rgbtBlue, rgbtGreen, rgbtRed*. Они идут именно в таком порядке. Структура выглядит так:

|  |
| --- |
| typedef struct RGBTRIPLE  {  BYTE rgbtBlue;  BYTE rgbtGreen;  BYTE rgbtRed;  } RGBTRIPLE; |

**1.5. Pragma.**

Структура BMFILEHEADER должна занимать 14 байт, а BMPINFOHEADER – 40, но из-за выравнивания данные структуры занимают большее количество байт. Для решения данной проблемы требуется отменить выравнивание с помощью директивы **#pragma.**

#pragma – определяемая реализацией директива, которая позволяет передавать компилятору различные иструкции.

#pragma pack(push, 2)// устанавливает выравнивание в 2 байтa

#pragma pack(pop)//возвращает предыдущую настройку

**1.6. Типы данных.**

Здесь используются типы данных из таблицы WinAPI. Проведем аналогию с типами данных языка С:

typedef unsigned short int WORD;

typedef unsigned long int DWORD;

typedef long int LONG;

typedef char BYTE;

**2. ПРОВЕРКА НА КОРРЕКТНОСТЬ ВВЕДЕННЫХ ДАННЫХ**

**2.1. Проверка на корректность координат**

В функции main реализована проверка на корректность координат с помощью следующего фрагмента кода:

|  |
| --- |
| if((x1-x0)<=0 && (y1-y0)<=0)  {  printf("Fail with coordinates");  return 0;  }  if((x1-x0)%2 == 0 || (y1-y0)%2 == 0)  {  printf("Fail with coordinates");  return 0;  } |

В данном фрагменте кода выполняется проверка корректности введенных координат а также проверка того, что заданная область имеет четное количество пикселей в длину и ширину. Это нужно для того, чтобы заданную область можно было корректно поделить на 4 равные части.

**2.2. Проверка существования файла.**

|  |
| --- |
| FILE \*bmp1;  FILE \*bmp2;  bmp1 = fopen(bmp, "rb"); //открытие файла для чтения  if(bmp1 == NULL)  {  printf("Fail with file opening\n"); //проверка на корректное открытие файла  return 0;  }  bmp2 = fopen("file2.bmp", "wb");  if(bmp2 == NULL)  {  printf("Fail with file opening\n"); //проверка на корректное открытие файла  return 0;  } |

В этом фрагменте кода приводится попытка открытия для чтения файла bmp1 и попытка создания и открытия для записи файла bmp2.

Если файл существует и мы обладаем достаточными правами для его открытия или создания, тогда выполняется функция, иначе – вывод сообщения об ошибки и завершение программы.

**2.3 Функция swap**

Функция swap делит заданную область BMP-файла на 4 части и меняет их местами по диагонали. Функция получает на вход координаты левого верхнего и правого нижнего углов заданной области. А также имя самого BMP-файла.

Сначала открывается исходный BMP-файл и создается новый BMP-файл в который будет помещено измененное изображение. Затем заголовки из исходного BMP-файла копируются в новый BMP-файл. Далее создается двумерный массив структур для работы с отдельными пикселями изображения.

После чего, с помощью цикла происходит запись цвета каждого пикселя исходного BMP-файла в двумерный массив структур, но уже в другом порядке.

При чтении цвета пикселя из области 1 его цвет записывается в двумерный массив но уже на место пикселя, из области 3 (рис. 2). Аналогично с пикселями из второй и четвертой областей.

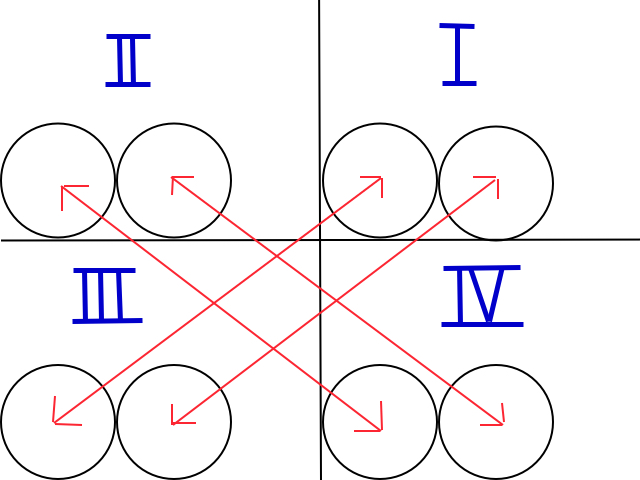


Рис. 2

Далее происходит запись данных с учетом выравнивания из двумерного массива структур в новый BMP-файл. После этого функция завершает свою работу.

|  |
| --- |
| int swap(int x0, int y0, int x1, int y1, char \*bmp)  {  BITMAPFILEHEADER BITMAP\_FILE;  BITMAPINFOHEADER BITMAP\_INFO;  RGBTRIPLE RGB;  COORDINATES\*\* cord;  FILE \*bmp1;  FILE \*bmp2;  bmp1 = fopen(bmp, "rb"); //открытие файла для чтения  if(bmp1 == NULL)  {  printf("Fail with file opening\n"); //проверка на корректное открытие файла  return 0;  }  bmp2 = fopen("file2.bmp", "wb");  if(bmp2 == NULL)  {  printf("Fail with file opening\n"); //проверка на корректное открытие файла  return 0;  }  fread(&BITMAP\_FILE, sizeof(BITMAP\_FILE), 1, bmp1); //копирование заголовков  fwrite(&BITMAP\_FILE, sizeof(BITMAP\_FILE), 1, bmp2);  fread(&BITMAP\_INFO, sizeof(BITMAP\_INFO), 1, bmp1);  fwrite(&BITMAP\_INFO, sizeof(BITMAP\_INFO), 1, bmp2);  int width = BITMAP\_INFO.biWidth;  int height = BITMAP\_INFO.biHeight;  cord=(COORDINATES\*\*)malloc(height\*sizeof(COORDINATES\*)); //создание двумерного массива структур  for(int i=0;i<height;i++)  cord[i]=(COORDINATES\*)malloc(width\*sizeof(COORDINATES));  size\_t padding = 0; //выравнивание  if ((width \* 3) % 4)  padding = 4 - (width \* 3) % 4;  for(int i=height-1; i>=0; i--)  {  for (int j=0; j<(width); j++)  {  fread(&RGB, sizeof(RGB),1, bmp1);  if(i >= y0 && i <= y0+(y1-y0)/2 && j >= x0 && j <= x0 + (x1-x0)/2)  {  cord[i+(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtBlue = RGB.rgbtBlue;  cord[i+(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtGreen = RGB.rgbtGreen;  cord[i+(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtRed = RGB.rgbtRed;  }  if(j > (x0+(x1-x0)/2) && j<=x1 && i >= y0 && i<=y0+(y1-y0)/2)  {  cord[i+(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtBlue = RGB.rgbtBlue;  cord[i+(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtGreen = RGB.rgbtGreen;  cord[i+(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtRed = RGB.rgbtRed;  }  if(i > y0+(y1-y0)/2 && i<=y1 && j>=x0 && j <= x0+(x1-x0)/2)  {  cord[i-(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtBlue = RGB.rgbtBlue;  cord[i-(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtGreen = RGB.rgbtGreen;  cord[i-(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtRed = RGB.rgbtRed;  }  if(i <= y1 && i > y0+(y1-y0)/2 && j <= x1 && j > x0+(x1-x0)/2)  {  cord[i-(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtBlue = RGB.rgbtBlue;  cord[i-(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtGreen = RGB.rgbtGreen;  cord[i-(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtRed = RGB.rgbtRed;  }  if(i<y0 || i>y1 || j>x1 || j<x0)  {  cord[i][j].rgbtBlue = RGB.rgbtBlue;  cord[i][j].rgbtGreen = RGB.rgbtGreen;  cord[i][j].rgbtRed = RGB.rgbtRed;  }  }  for(int k = 0; k<padding; k++)  {  getc(bmp1);  }  }  for(int i=height-1; i>=0; i--)  {  for(int j=0; j<width; j++)  fwrite(&cord[i][j], sizeof(RGB), 1, bmp2);  for(int k = 0; k<padding; k++)  {  fprintf(bmp2, "%c", 0);  }  }  fclose(bmp1);  fclose(bmp2);  return 0;  } |

**2.4 Функция main**

В функции main происходит считывание координат и имени BMP-файла, проверка координат на корректность и вызов функции swap.

|  |
| --- |
| int main()  {  int x0, y0, x1, y1;  char bmp[30];  fgets(bmp, 30, stdin);  bmp[strlen(bmp)-1] = '\0';  scanf("%d%d%d%d", &x0, &y0, &x1, &y1);  if((x1-x0)<=0 && (y1-y0)<=0)  {  printf("Fail with coordinates");  return 0;  }  if((x1-x0)%2 == 0 || (y1-y0)%2 == 0)  {  printf("Fail with coordinates");  return 0;  }  swap(x0, y0, x1, y1, bmp);  return 0;  } |

**3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ**

**ПРИМЕР 1.**

**Исходное изображение:**



**Входные значения:**

file1.bmp

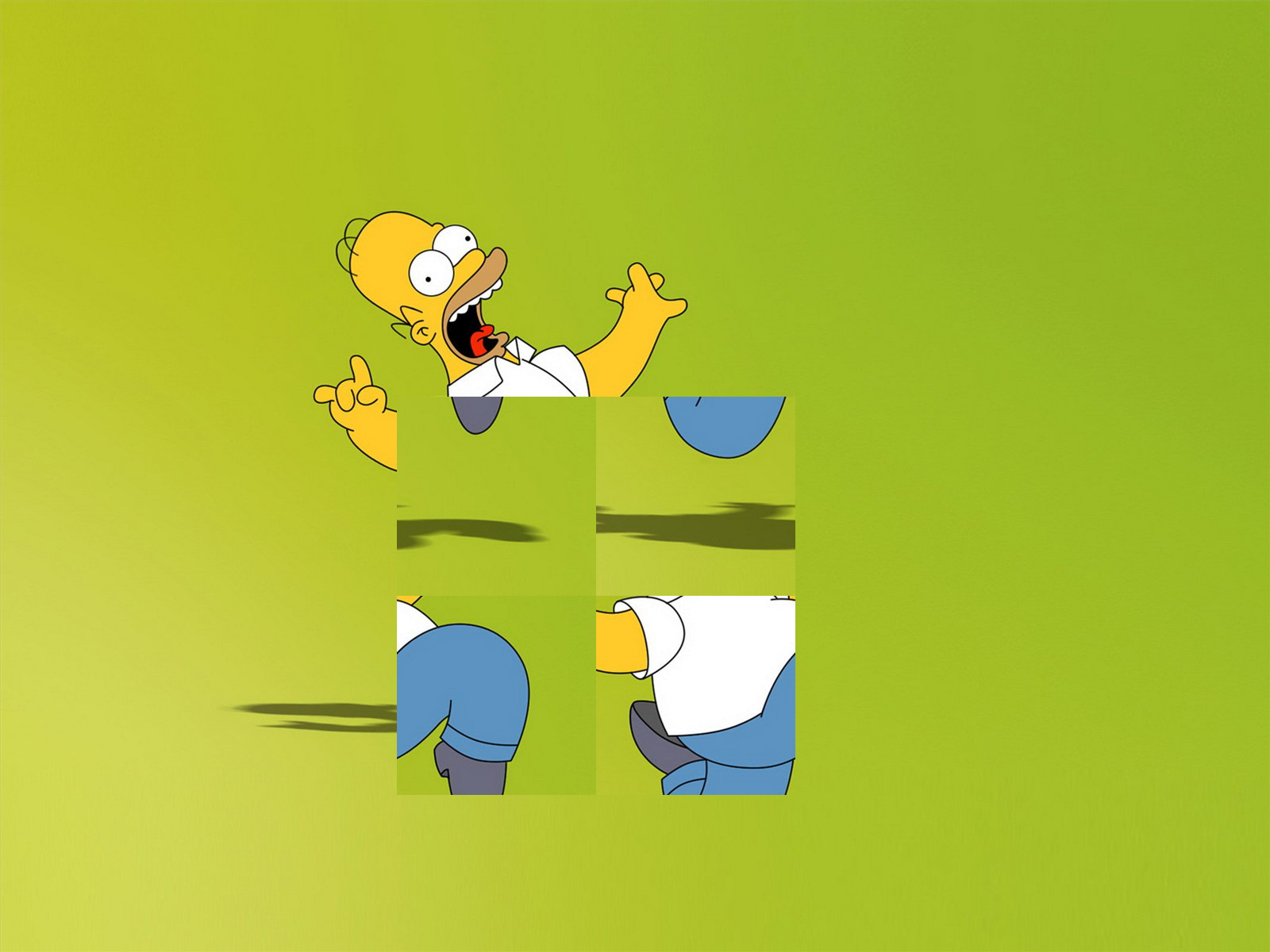
x0 = 500

y0 = 500

x1 = 1001

y1 = 1001

**Результат:**



**ПРИМЕР 2.**

**Исходное изображение:**

****

**Входные данные:**

file2.bmp

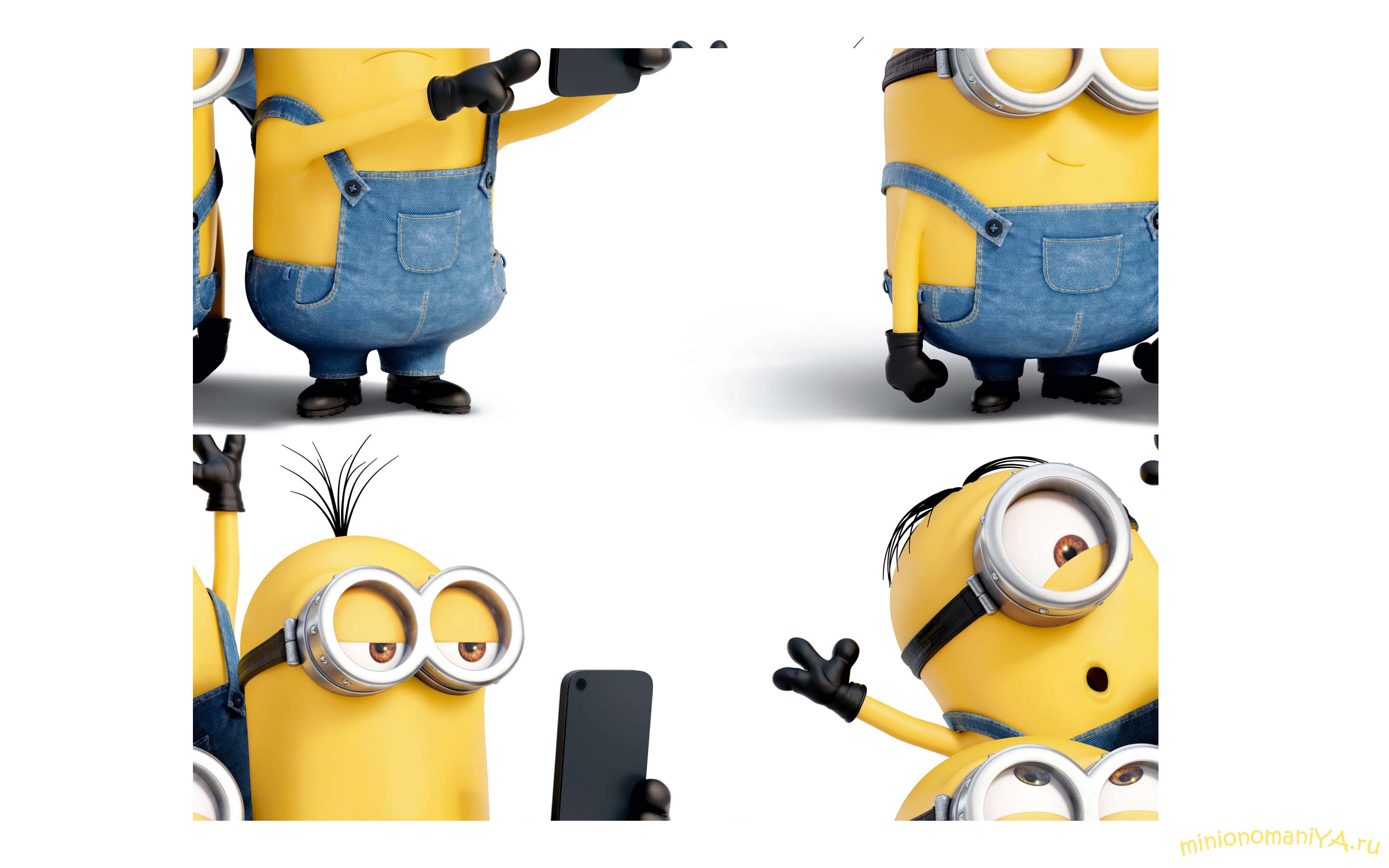
x0 = 400

y0 = 100

x1 = 2401

y1 = 1701

**Результат:**



**ЗАКЛЮЧЕНИЕ**

В ходе работы была реализована программа на языке программирования С, которая делит заданную область BMP-файла на 4 равные части и меняет их местами по диагонали. Приведено полное описание исходного кода. Код и отчет по работе были загружены в репозиторий на Git.

**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 А**

**Исходный код программы**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #pragma pack(push, 2)  typedef unsigned short WORD;  typedef unsigned int DWORD;  typedef int LONG;  typedef unsigned char BYTE;  typedef struct COORDINATES  {  BYTE rgbtBlue;  BYTE rgbtGreen;  BYTE rgbtRed;  }COORDINATES;  typedef struct tagBITMAPFILEHEADER  {  WORD bfType;  DWORD bfSize;  WORD bfReserved1;  WORD bfReserved2;  DWORD bfOffBits;  } BITMAPFILEHEADER, \*PBITMAPFILEHEADER;  typedef struct tagBITMAPINFOHEADER  {  DWORD biSize;  LONG biWidth;  LONG biHeight;  WORD biPlanes;  WORD biBitCount;  DWORD biCompression;  DWORD biSizeImage;  LONG biXPelsPerMeter;  LONG biYPelsPerMeter;  DWORD biClrUsed;  DWORD biClrImportant;  } BITMAPINFOHEADER, \*PBITMAPINFOHEADER;  typedef struct RGBTRIPLE  {  BYTE rgbtBlue;  BYTE rgbtGreen;  BYTE rgbtRed;  } RGBTRIPLE;  #pragma pack(pop)  int swap(int x0, int y0, int x1, int y1, char \*bmp)  {  BITMAPFILEHEADER BITMAP\_FILE;  BITMAPINFOHEADER BITMAP\_INFO;  RGBTRIPLE RGB;  COORDINATES\*\* cord;  FILE \*bmp1;  FILE \*bmp2;  bmp1 = fopen(bmp, "rb"); //открытие файла для чтения  if(bmp1 == NULL)  {  printf("Fail with file opening\n"); //проверка на корректное открытие файла  return 0;  }  bmp2 = fopen("file2.bmp", "wb");  if(bmp2 == NULL)  {  printf("Fail with file opening\n"); //проверка на корректное открытие файла  return 0;  }  fread(&BITMAP\_FILE, sizeof(BITMAP\_FILE), 1, bmp1); //копирование заголовков  fwrite(&BITMAP\_FILE, sizeof(BITMAP\_FILE), 1, bmp2);  fread(&BITMAP\_INFO, sizeof(BITMAP\_INFO), 1, bmp1);  fwrite(&BITMAP\_INFO, sizeof(BITMAP\_INFO), 1, bmp2);  int width = BITMAP\_INFO.biWidth;  int height = BITMAP\_INFO.biHeight;  cord=(COORDINATES\*\*)malloc(height\*sizeof(COORDINATES\*)); //создание двумерного массива структур  for(int i=0;i<height;i++)  cord[i]=(COORDINATES\*)malloc(width\*sizeof(COORDINATES));  size\_t padding = 0; //выравнивание  if ((width \* 3) % 4)  padding = 4 - (width \* 3) % 4;  for(int i=height-1; i>=0; i--)  {  for (int j=0; j<(width); j++)  {  fread(&RGB, sizeof(RGB),1, bmp1);  if(i >= y0 && i <= y0+(y1-y0)/2 && j >= x0 && j <= x0 + (x1-x0)/2)  {  cord[i+(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtBlue = RGB.rgbtBlue;  cord[i+(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtGreen = RGB.rgbtGreen;  cord[i+(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtRed = RGB.rgbtRed;  }  if(j > (x0+(x1-x0)/2) && j<=x1 && i >= y0 && i<=y0+(y1-y0)/2)  {  cord[i+(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtBlue = RGB.rgbtBlue;  cord[i+(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtGreen = RGB.rgbtGreen;  cord[i+(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtRed = RGB.rgbtRed;  }  if(i > y0+(y1-y0)/2 && i<=y1 && j>=x0 && j <= x0+(x1-x0)/2)  {  cord[i-(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtBlue = RGB.rgbtBlue;  cord[i-(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtGreen = RGB.rgbtGreen;  cord[i-(y1+1-y0)/2][j+(x1+1-x0)/2].rgbtRed = RGB.rgbtRed;  }  if(i <= y1 && i > y0+(y1-y0)/2 && j <= x1 && j > x0+(x1-x0)/2)  {  cord[i-(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtBlue = RGB.rgbtBlue;  cord[i-(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtGreen = RGB.rgbtGreen;  cord[i-(y1+1-y0)/2][j-(x1+1-x0)/2].rgbtRed = RGB.rgbtRed;  }  if(i<y0 || i>y1 || j>x1 || j<x0)  {  cord[i][j].rgbtBlue = RGB.rgbtBlue;  cord[i][j].rgbtGreen = RGB.rgbtGreen;  cord[i][j].rgbtRed = RGB.rgbtRed;  }  }  for(int k = 0; k<padding; k++)  {  getc(bmp1);  }  }  for(int i=height-1; i>=0; i--)  {  for(int j=0; j<width; j++)  fwrite(&cord[i][j], sizeof(RGB), 1, bmp2);  for(int k = 0; k<padding; k++)  {  fprintf(bmp2, "%c", 0);  }  }  fclose(bmp1);  fclose(bmp2);  return 0;  }  int main()  {  int x0, y0, x1, y1;  char bmp[30];  fgets(bmp, 30, stdin);  bmp[strlen(bmp)-1] = '\0';  scanf("%d%d%d%d", &x0, &y0, &x1, &y1);  if((x1-x0)<=0 && (y1-y0)<=0)  {  printf("Fail with coordinates");  return 0;  }  if((x1-x0)%2 == 0 || (y1-y0)%2 == 0)  {  printf("Fail with coordinates");  return 0;  }  swap(x0, y0, x1, y1, bmp);  return 0;  } |