|  |  |
| --- | --- |
| Федеральное агентство железнодорожного транспорта  Омский государственный университет путей сообщения  Кафедра «Автоматика и системы управления»  РАСТРОВЫЕ РИСУНКИ  Лабораторная работа № 8  по дисциплине «Информационные технологии» | |
|  | Студент гр. 22м                            Н.И. Яковлев  «    »                  2022 г.  Руководитель –  старший преподаватель кафедры «АиСУ»                            Т. В. Васеева  «    »                  2022 г. |
| Омск 2022 | |

Цель работы

Ознакомиться с форматом для хранения растровых изображений «bmp» и его структурой.

# Выполнение работы

## Предустановки к работе

Для начала необходимо открыть наш .bmp файл в средстве просмотра фотографий Windows (рисунок 1), а также в редакторе frhead (рисунок 2).

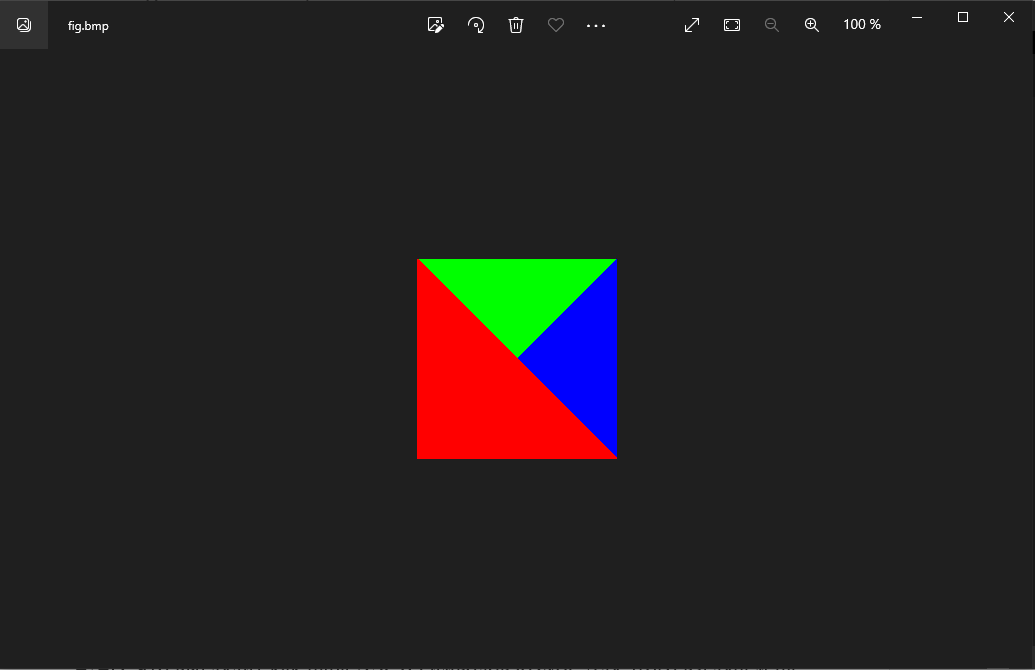


Рисунок  – «fig.bmp» в средстве просмотра фотографий Windows

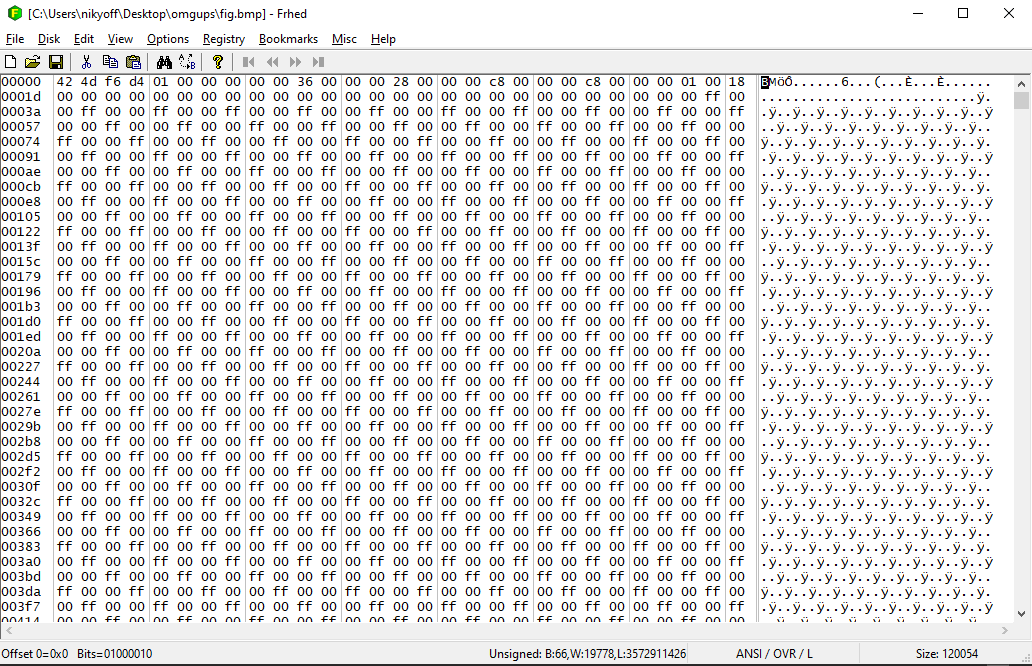


Рисунок  – «fig.bmp» в редакторе frhead

## Структура «BITMAPFILEHEADER»

Данная структура представлена в таблице 1.

Таблица 1 – Структура «BITMAPFILEHEADER»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поз.  (byte) | Размер (байты) | Имя | Тип WinAPI | Описание |
| 0 | 2 | bfType | WORD | Отметка для отличия формата от других (сигнатура формата) |
| 2 | 4 | bfSize | DWORD | Размер файла в байтах |
| 6 | 2 | bfReserved1 | WORD | Зарезервированы и должны содержать ноль |
| 8 | 2 | bfReserved2 | WORD |
| 10 | 4 | bfOffBits | DWORD | Положение пиксельных данных относительно начала данной структуры (в байтах) |

Убедимся в том, что в «bfType» записано единственно возможное значение – 424D, что выглядит как пара ASCII символов «BM». Для этого взглянем на рисунок 3.

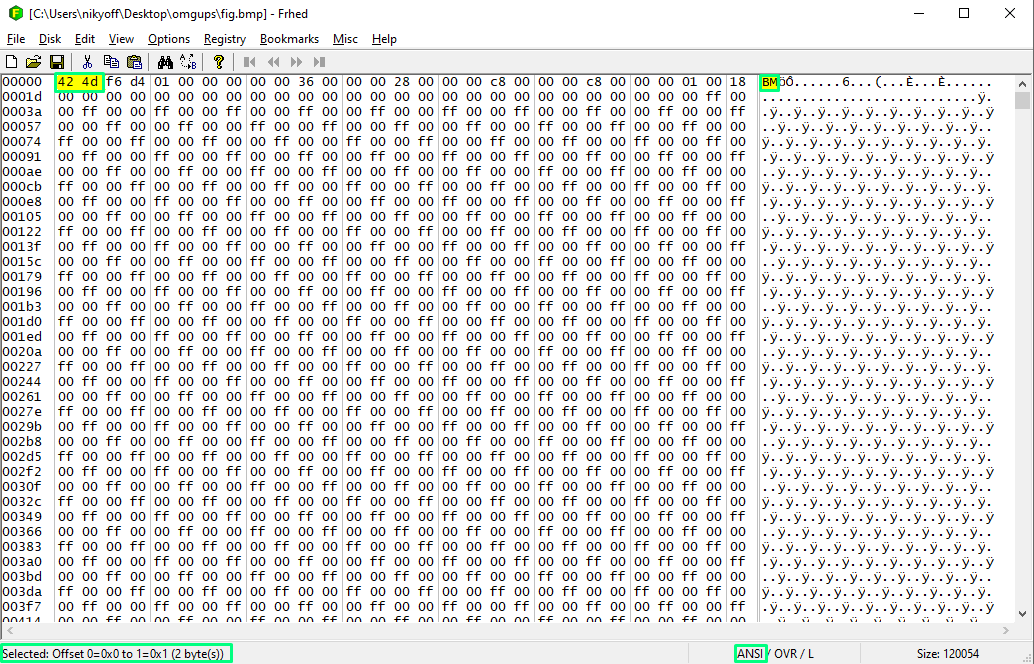


Рисунок  – «bfType»

Затем следует сравнить «bfSize» с размеров файла сообщаемым операционной системой. Обратимся к рисунку 4.

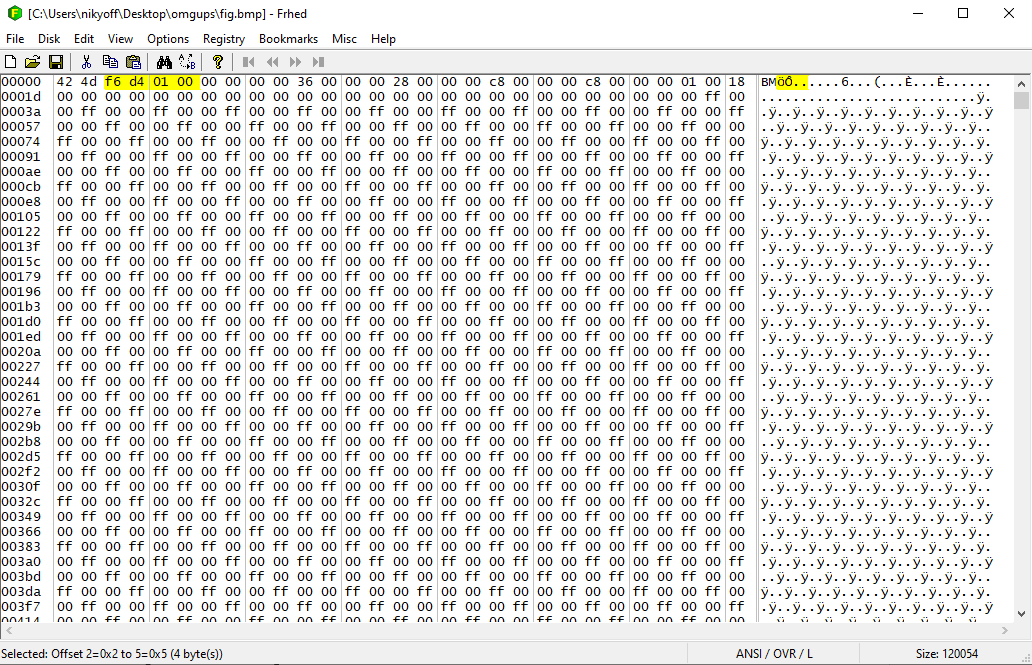


Рисунок  – «bfSize»

Исходный порядок байт «f6 d4 01 00» следует интерпретировать в «little-endian», следовательно значение bfSize примет вид «00 01 d4 f6», затем следует перевести шестнадцатеричное значение в десятичное. В результате перевода получим 120054 (рисунок 5).

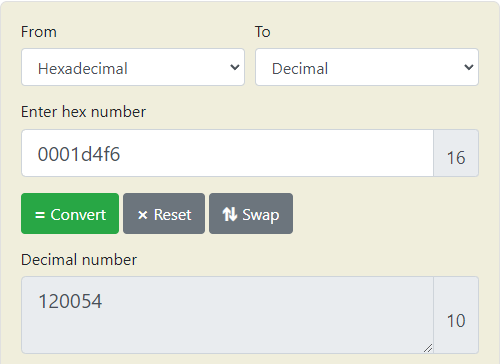


Рисунок  – Перевод систем счисления

Затем, следует обратиться к системному значению размера исходного файла (рисунок 6).

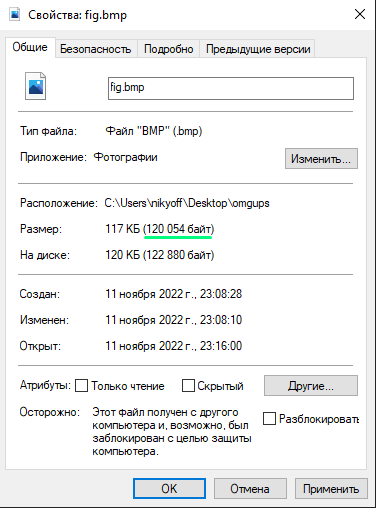


Рисунок  – Системный размер исходного файла

Системный размер исходного файла равен 120054 байтам, что в свою очередь соответствует полученному нами значению в поле «bfSize».

Зарезервированные поля не принимают участия в данной работе, потому перейдем к полю «bfOffBits». Для наглядности обратимся к рисунку 7.

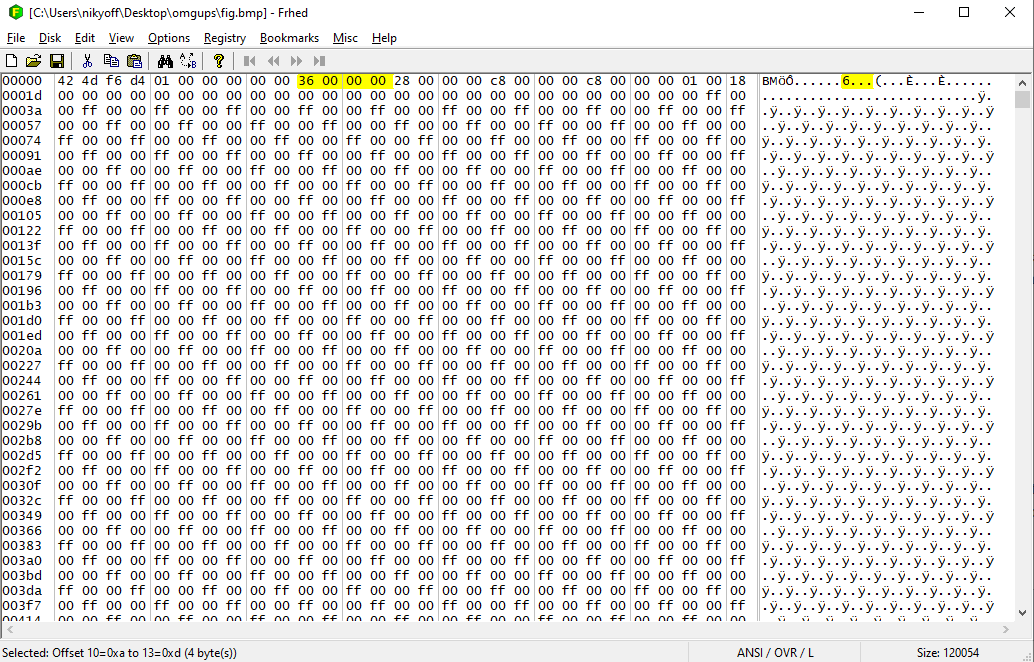


Рисунок  – «bfOffBits»

В результате получим последовательность байт «36 00 00 00», интерпретируем ее также в «little-endian», то есть исходная последовательность байт равна «00 00 00 36», что равняется в десятичной системе счисления 54 (рисунок 8).

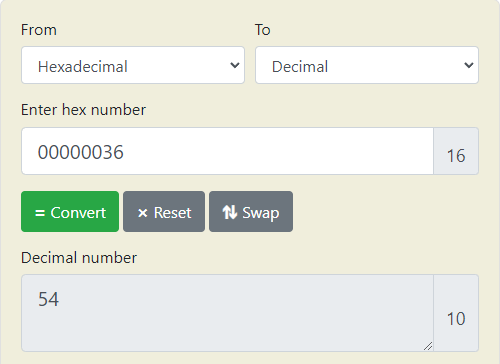


Рисунок  – Перевод систем счисления

## Структура «BITMAPINFO»

Данная структура зависит от версии, поэтому для начала необходимо определить ее. Всего выделяют четыре версии: «core», «3», «4» и «5». Для определения версии требуется узнать значение первых четырех байт (DWORD), значение которых определяет как размер структуры, так и саму версию. Для наглядности обратимся к рисунку 9.

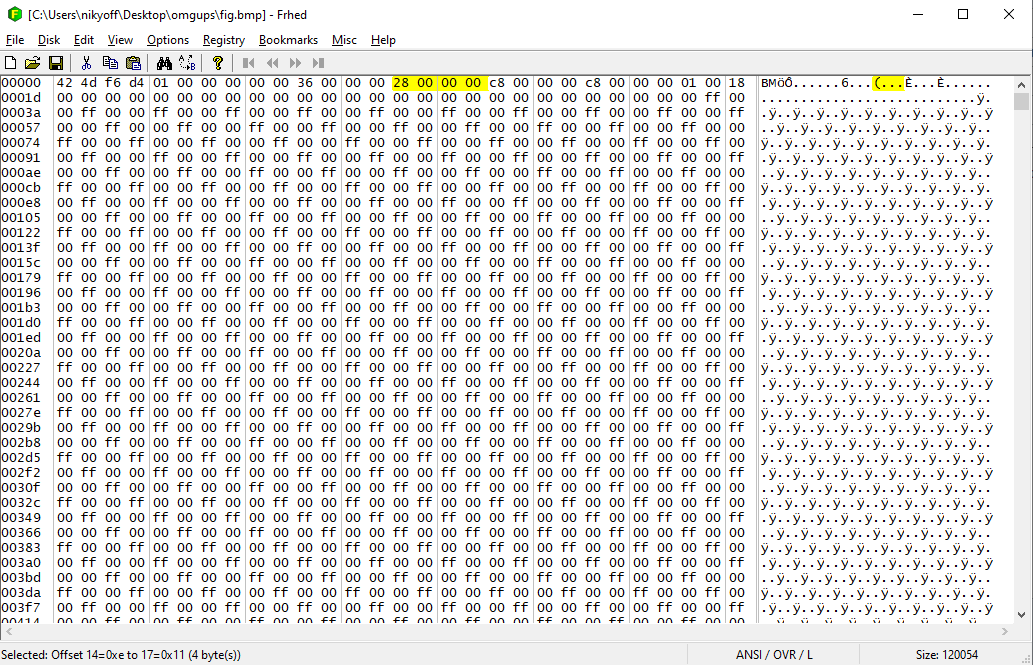


Рисунок  – Определение версии

В результате получаем следующую последовательность байт «28 00 00 00», ее следует интерпретировать в little-endian. В результате исходная последовательность байт будет «00 00 00 28», переводя в десятичную систему счисления, получим 40 (рисунок 10).

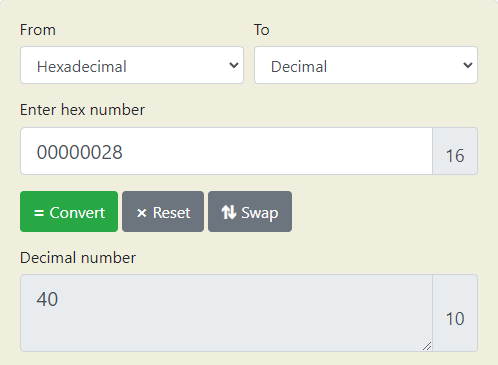


Рисунок  – Перевод систем счисления

Соотношение значения и версий приведено в таблице 2.

Таблица 2 – Соотношение версией и размера

|  |  |
| --- | --- |
| Версия | Размер (байты) |
| CORE | 12 |
| 3 | 40 |
| 4 | 108 |
| 5 | 124 |

Таким образом, структура имеет третью версию.

Исходя из версии, можно провести обзор структуры, для этого обратимся к таблице 3.

Таблица 3 – Структура «BITMAPINFO» третьей версии

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Позиция  в файле  (byte) | Размер  (байты) | Имя | Тип WinAPI | Описание |
| 14 | 4 | biSize | DWORD | Размер данной структуры в байтах, указывающий также на версию структуры |
| 18 | 4 | biWidth | LONG | Ширина растра в пикселях |
| 22 | 4 | biHeight | LONG | * высота растра в пикселях * порядок следования строк в двумерных массивах |
| 26 | 2 | biPlanes | WORD | Это поле используется в значках и курсорах Windows. |

Продолжение таблицы 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Позиция  в файле  (hex) | Размер  (байты) | Имя | Тип WinAPI | Описание |
| 28 | 2 | biBitCount | WORD | Количество бит на пиксель |
| 30 | 4 | biCompression | DWORD | Указывает на способ хранения пикселей |
| 34 | 4 | biSizeImage | DWORD | Размер пиксельных данных в байтах или ноль, в случае хранения в виде двумерного массива |
| 38 | 4 | biXPelsPerMeter | LONG | Количество пикселей на метр по горизонтали и вертикали |
| 42 | 4 | biYPelsPerMeter | LONG |
| 46 | 4 | biClrUsed | DWORD | Размер таблицы цветов в ячейках |
| 50 | 4 | biClrImportant | DWORD | Количество ячеек от начала таблицы цветов до последней используемой |

Для определения количества байт, занимаемых пиксельными данными нас интересуют поля «biWidth», «biHeight» и «biBitCount».

Обратимся к рисункам 11 – 13 для того, чтобы наглядно показать соответствующе поля.

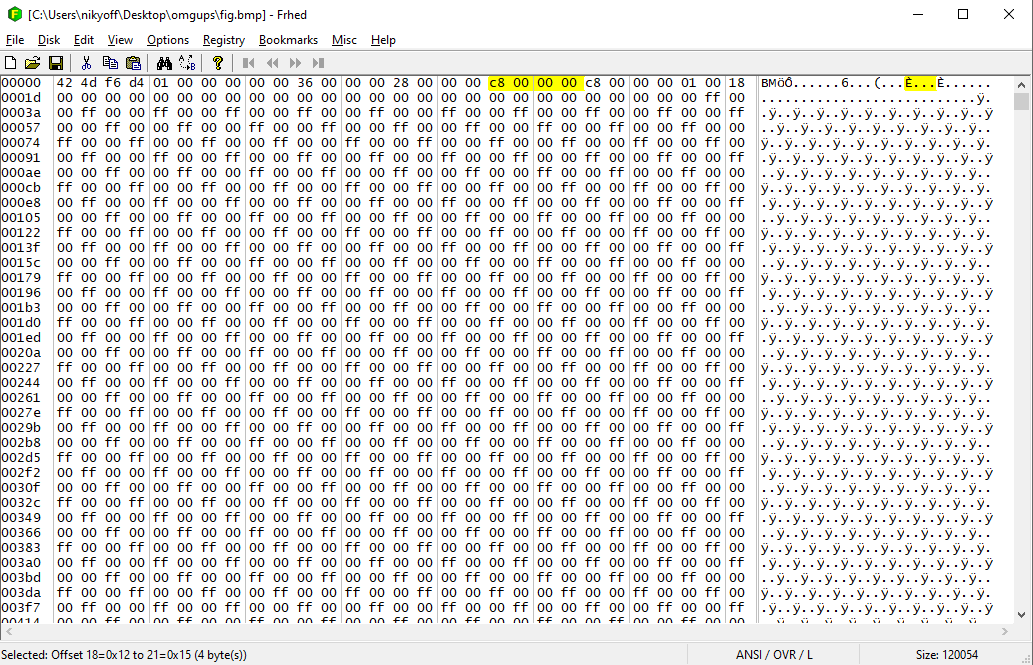


Рисунок  – «biWidth»

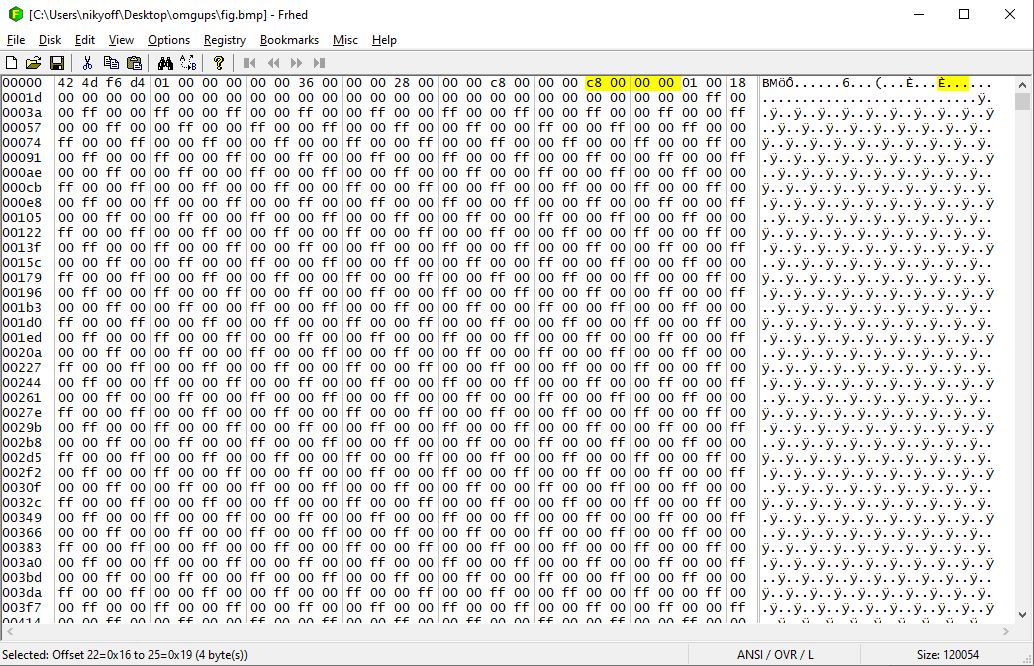


Рисунок  – «biHeight»

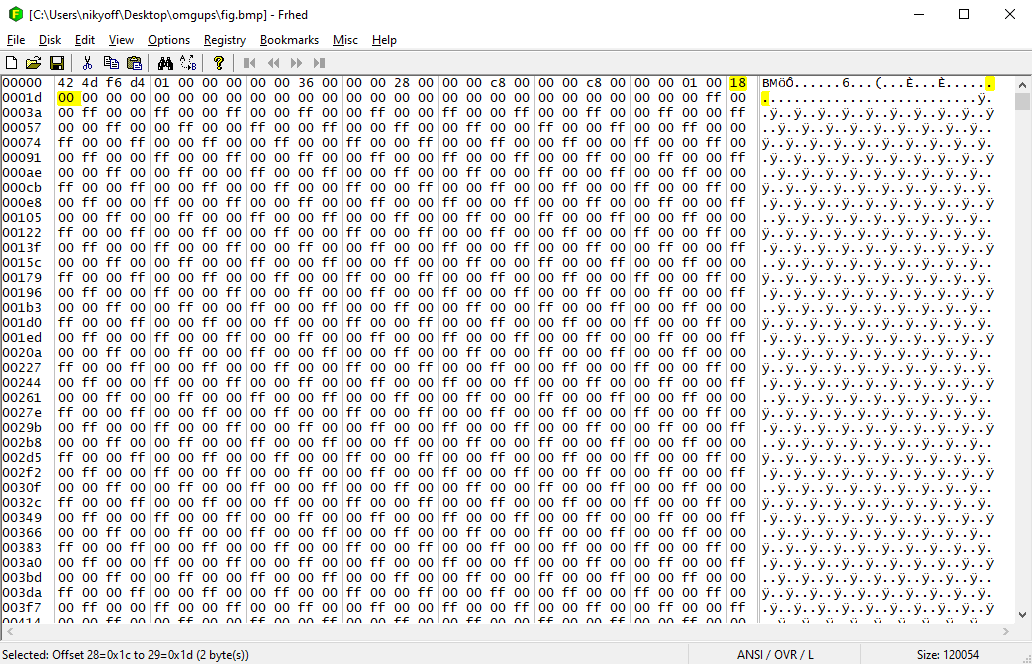


Рисунок  – «biBitCount»

Получаем следующие набор последовательностей байт:

biWidth: «c8 00 00 00»;

biHeight: «c8 00 00 00»;

biBitCount: «18 00».

Затем будем рассматривать их в little-endian последовательности:

biWidth: «00 00 00 c8»;

biHeight: «00 00 00 c8»;

biBitCount: «00 18».

Затем представим их в десятичной системе счисления (рисунки 14-16):

biWidth: 200;

biHeight: 200;

biBitCount: 24.

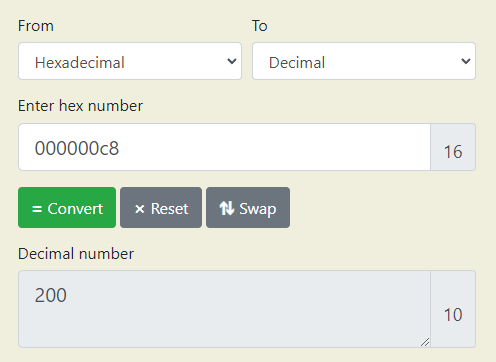


Рисунок  – Перевод систем счисления

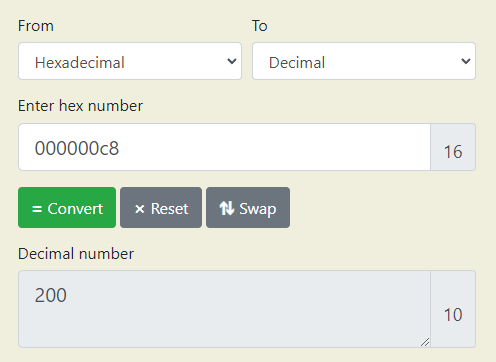


Рисунок  – Перевод систем счисления

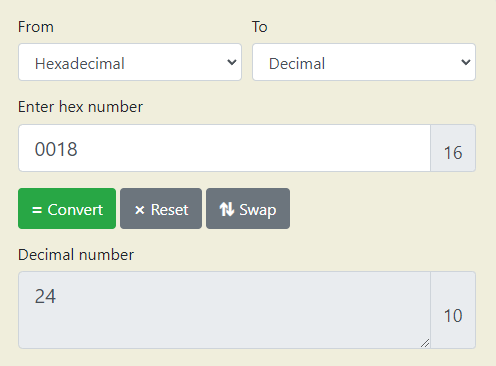


Рисунок  – Перевод систем счисления

Для нахождения количества байт, занимаемых пиксельными данными, необходимо перемножить три получившихся значения. В результате получим 960000 бит, что в байтовом эквиваленте соответствует числу 120000. Доказать истинность результата можно проведя небольшие вычисления. Структуры «BITMAPFILEHEADER» и «BITMAPINFO», в сумме занимают 54 байта, а общий размер файла равен 120054 байтам. Соответственно разница между этими значениями и образует количество байт, занимаемыми пиксельными данными.

## Сравнение

При открытии частичного представления, необходимо выбрать начальное смещение и количество загружаемых байт. Необходимо загрузить 120000 байт с смещением в 54 байта (x36 в hex), либо с конца. Для наглядности приведу примеры на рисунках 17 – 18.

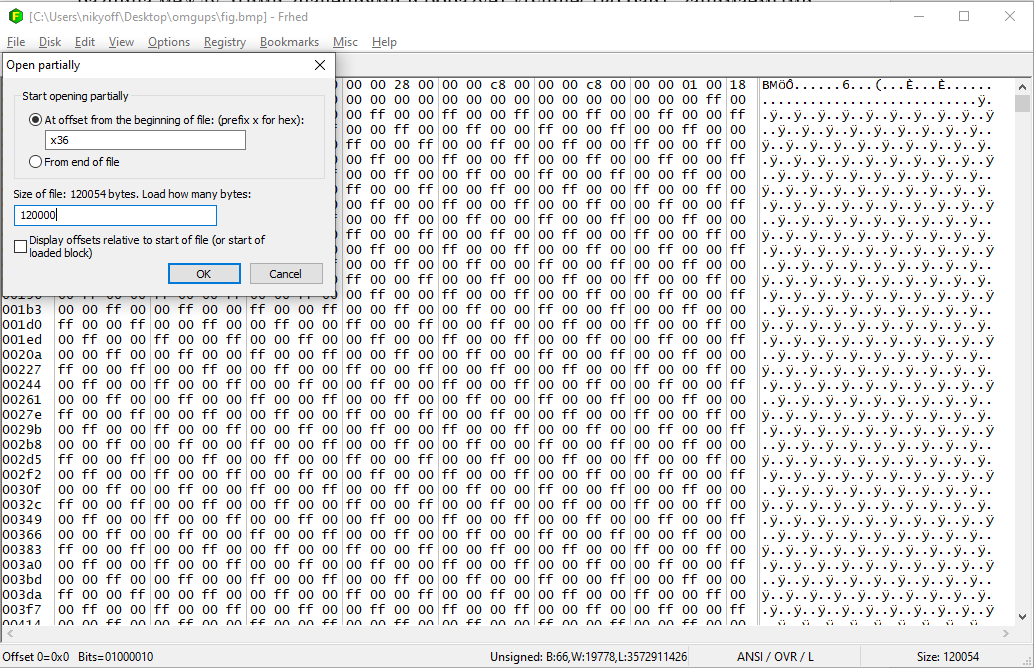


Рисунок  – Настройка частичного отображения со смещением

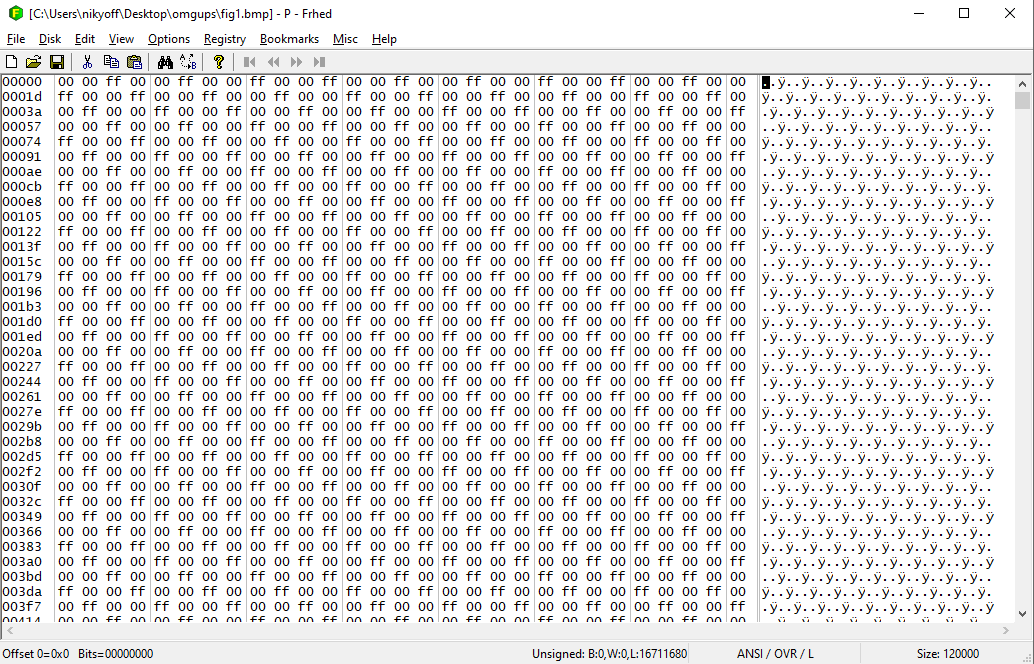


Рисунок  – Частичное отображения со смещением

Исходя из ранее полученных значений, мы можем говорить о том, что каждый пиксель закодирован 3 байтами, что соответствует модели rgb, в свою очередь rgb представляет из себя смешивание трех базовых цветов – red (красный), green (зеленый), blue (синий) и кодирует каждый из базовых цветов в одном байте.

Пиксели образуют картинку, начиная от нижнего левого пикселя и двигаясь вправо и вверх. Таким образом, первые три байта отвечают за rgb модель представления первого левого нижнего пикселя. Следовательно при последовательном изменении первых трех байт на 00 в результате получим черный цвет у первого пикселя (рисунок 19 – 21).

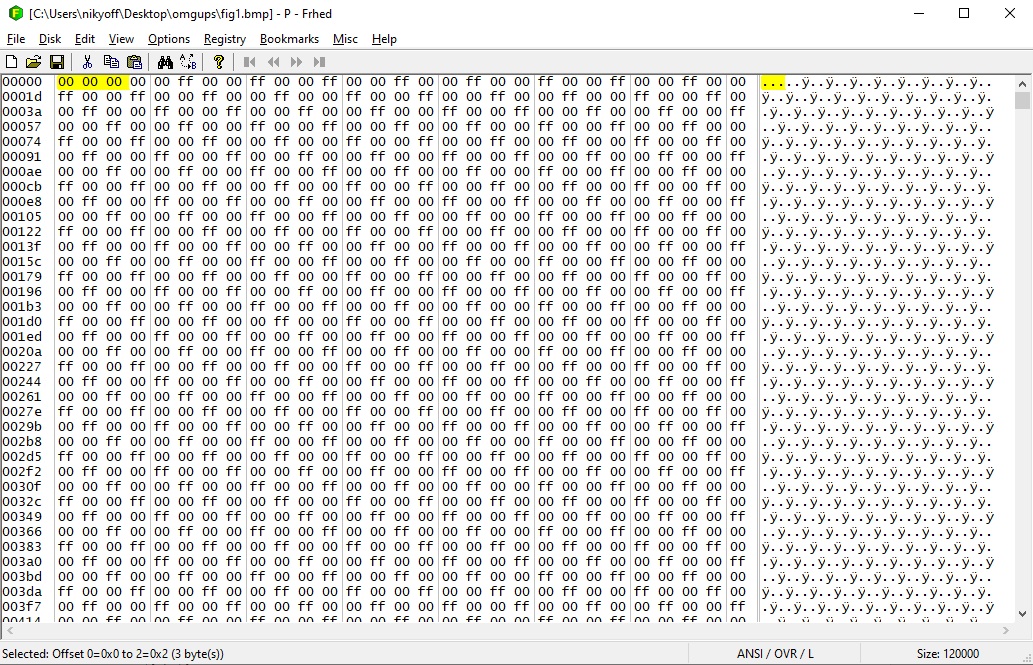


Рисунок  – Измененный набор байт первого пикселя

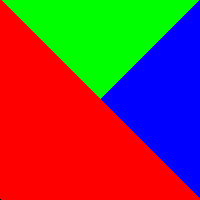


Рисунок  – Черный цвет пикселя



Рисунок  – Черный цвет пикселя в масштабе

Как и следовало ожидать, при изменении первых трех байт, изменяется цвет первого отображаемого пикселя на черный, так как «00 00 00» в rgb является черным цветом.

При изменении двух из трех байт на «ff», а третьего на «00», мы получим три различных результата:

1. «ff ff 00» – желтый цвет в результате смешивания красного и зеленого (рисунки 22 – 23);
2. «00 ff ff» – голубой цвет в результате смешивания зеленого и синего (рисунки 24 – 25);
3. «ff 00 ff» – лиловый цвет в результате смешивания красного и синего (рисунки 26 – 27).

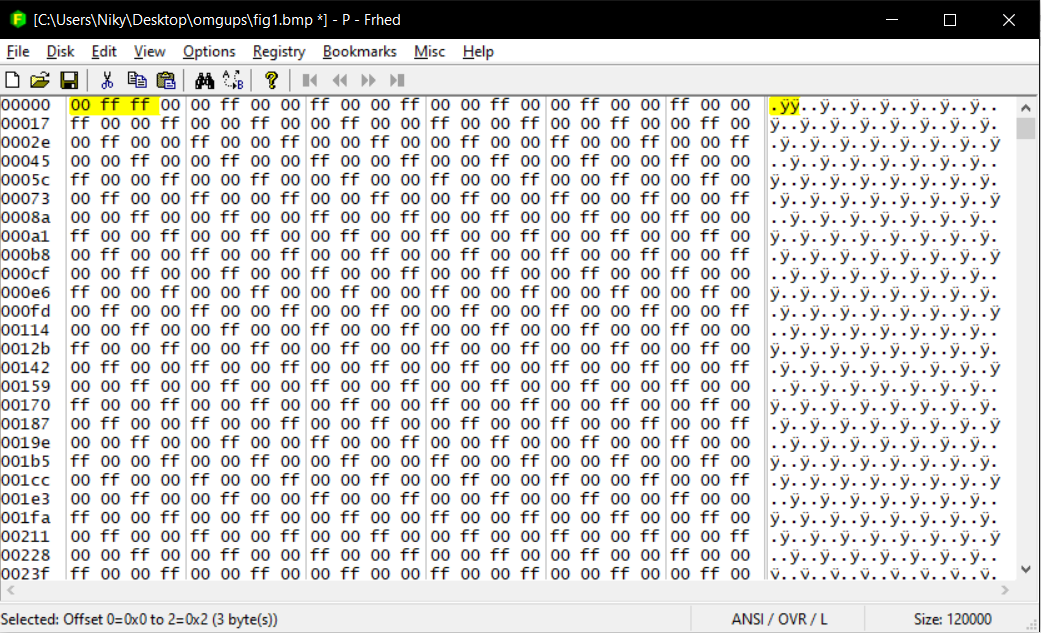
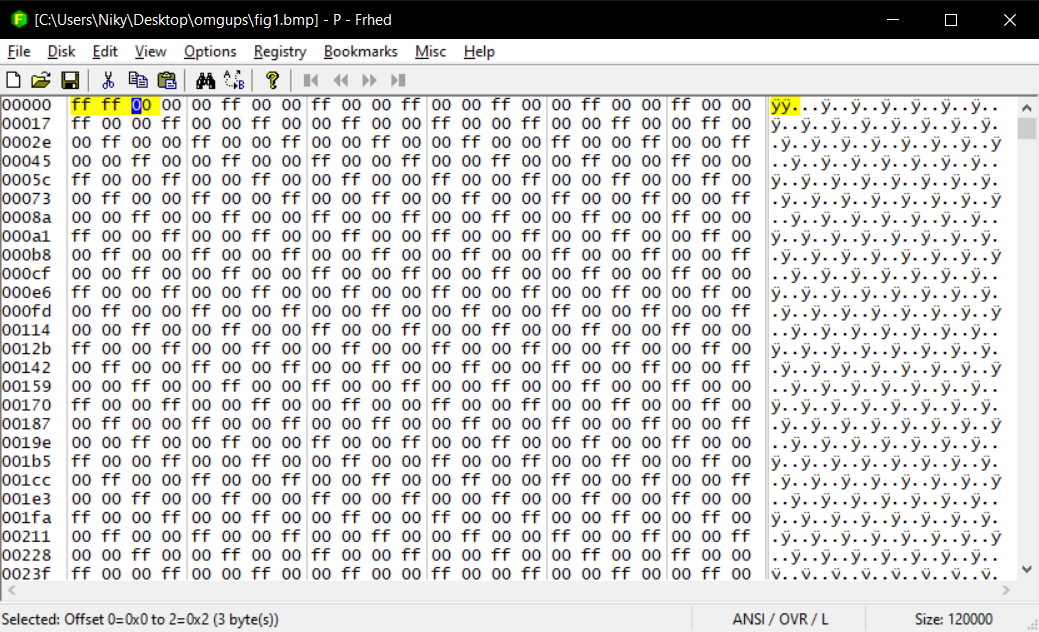


Рисунок  – «ff ff 00» в frhead



Рисунок  – «ff ff 00» в средстве просмотра фотографий Windows

Рисунок  – «00 ff ff» в frhead

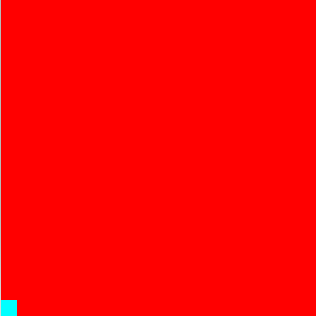


Рисунок  – «00 ff ff» в средстве просмотра фотографий Windows

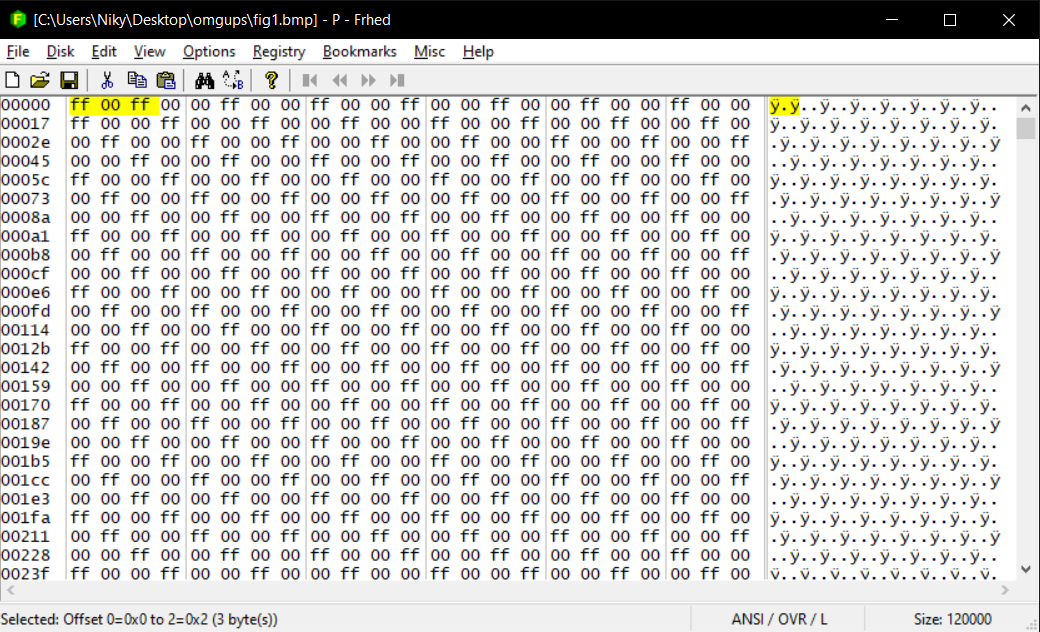


Рисунок  – «ff 00 ff» в frhead

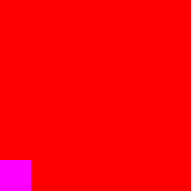


Рисунок  – «ff 00 ff» в средстве просмотра фотографий Windows

## Масштабирование растрового изображения

При масштабировании растрового изображения, нам будут видны все больше и больше пиксели, из которых и состоит само изображение. Рассмотреть наглядно можно на рисунках 27 – 28.

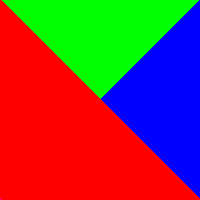


Рисунок  – Исходное изображение

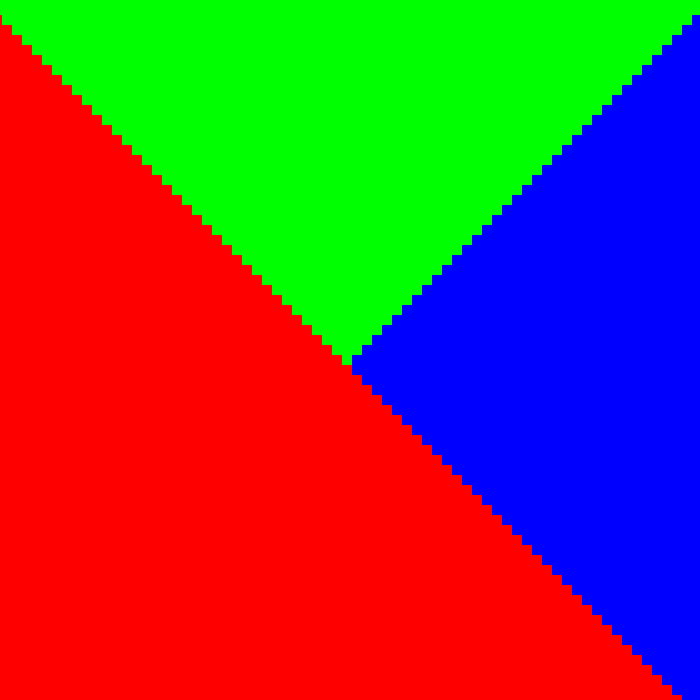


Рисунок  – Масштабированное изображение

Определяем алгоритм разложения отрезка (линии) в растр. Для этого рассмотрим рисунок 30.

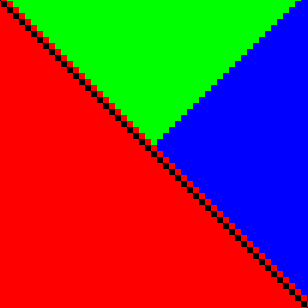


Рисунок  – Изображение с диагональной линией

Как видно, линия не имеет сглаживаний, следовательно в MS Paint используется алгоритм Брезенхэма.

## Сжатие

Соответствия между качеством сжатия и размером выходного файла приведены в таблице 2.

Таблица 2 – Соответствие качества сжатия и размера файла

|  |  |  |
| --- | --- | --- |
| Изображение | Качество сжатия | Размер файла (байт) |
|  | 100 | 696 |
|  | 80 | 696 |
|  | 60 | 692 |
|  | 40 | 696 |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
|  | 20 | 692 |
|  | 0 | 410 |

Искажение не заметно при качестве сжатия вплоть до 20.

Повторим те же действия для фотографии из второй лабораторной работы. Обратимся к таблице 3.

Таблица 3 – Соответствие качества сжатия и размера файла

|  |  |  |
| --- | --- | --- |
| Изображение | Качество сжатия | Размер файла (байт) |
|  | 100 | 4358 |
|  | 80 | 1873 |
|  | 60 | 1616 |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
|  | 40 | 1525 |
|  | 20 | 1385 |
|  | 0 | 697 |

Искажение не заметно при качестве сжатия вплоть до 20.

# Контрольные вопросы

## Модификация алгоритма Брезенхэма со сглаживанием границы

Модификация алгоритма Брезенхэма со сглаживанием границы – это модификация для рисования окружностей. Все работает по схожему принципу с основным алгоритмом. Происходит расчет для одного октанта, а все остальные части окружности дорисовываются по симметрии. Для наглядности обратимся к рисунку 31.

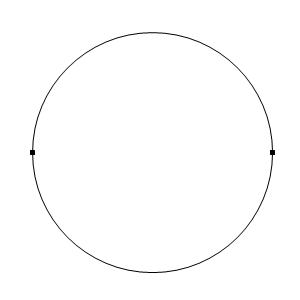


Рисунок  – Модификация алгоритма Брезенхэма

## Сравнение алгоритмов Брезенхэма и Ву

Сравнение приведено в таблице 4.

Таблица 4 – Сравнение алгоритмов

|  |  |  |
| --- | --- | --- |
| Параметр | Брезенхэм | Ву |
| Разложение отрезка в растр | Да | Да |
| Скорость | Высокая | Близкая к высокой |
| Сглаживание | Нет | Да |
| Рисование окружности | Да | Нет |
| Сложность алгоритма | O(n) | O(n) |

## Выполнить для другого рисунка BMP все задания с 1 по 10 из лабораторной

Задание будет выполняться для рисунка 32.



Рисунок  – Исходное изображение (в приближении)

Для начала откроем bmp файл в редакторе frhead (рисунок 33)

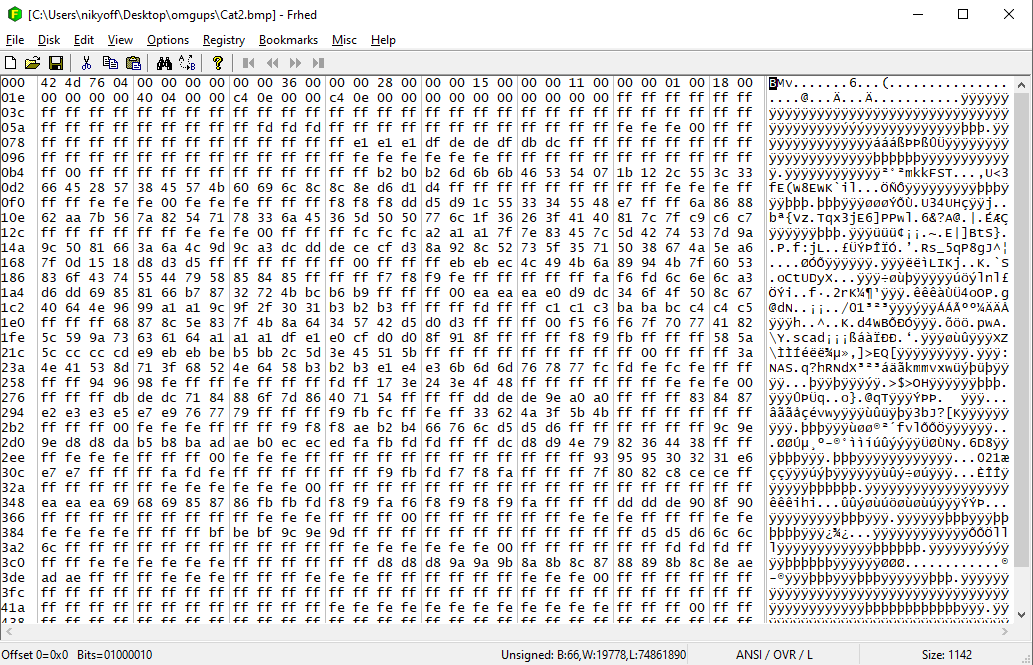


Рисунок  – «Cat2.bmp» в редакторе frhead

Для начала определим структуру «BITMAPFILEHEADER» для данного файла, сама структура приведена в таблице 1.

Первоначально селдует убедиться в том, что в «bfType» записано единственное возможное значение – 424D, должно отображаться как пара ACII символов «BM». Для этого взглянем на рисунок 34.

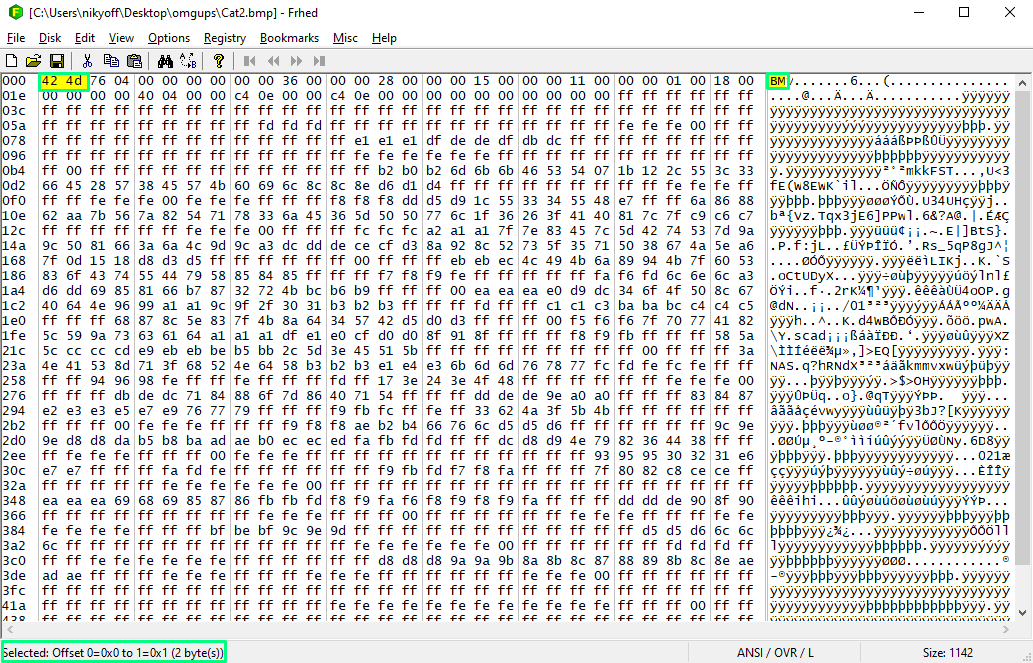


Рисунок  – «bfType»

Затем, сравним «bfSize» с размером файла сообщаемым операционной системой, для этого обратимся к рисунку 35.

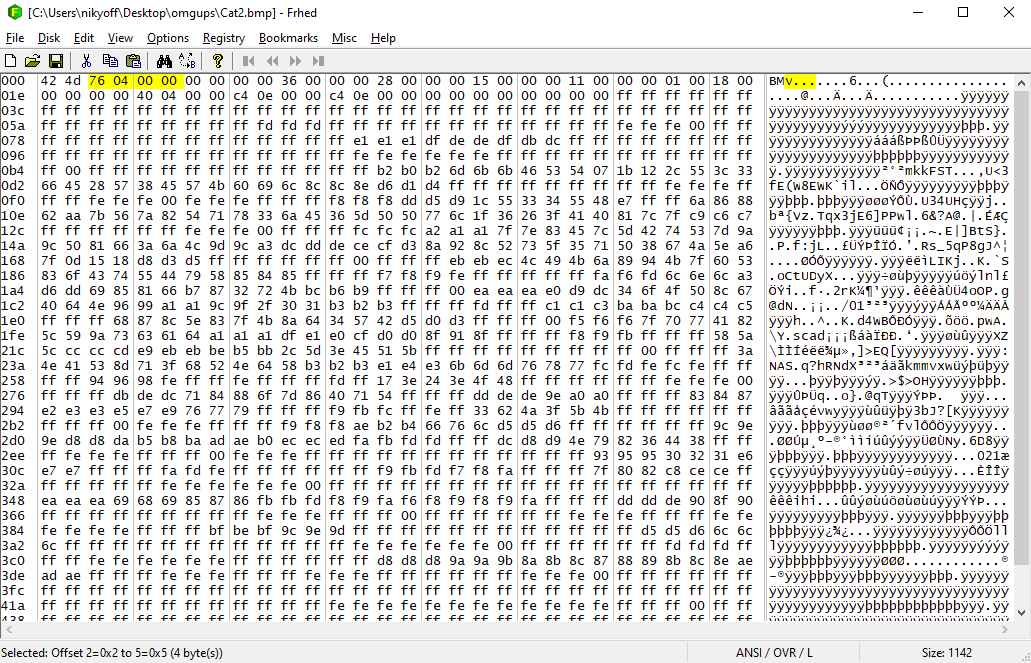


Рисунок  – «bfSize»

Исходный порядок байт «76 04 00 00» следует интерпретировать в little-endian, следовательно значение «bfSize» примет вид «00 00 04 76», затем следует перевести шестнадцатеричное значение в десятичное. В результате перевода получим 1142 (рисунок 36).

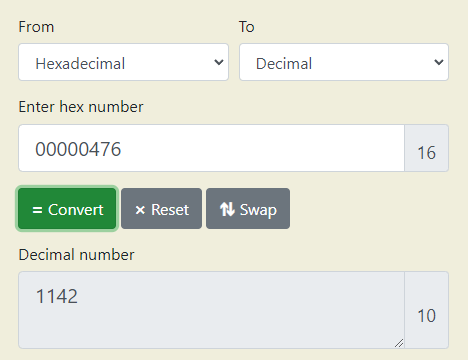


Рисунок  – Перевод систем счисления

Теперь обратимся к системному значению размера исходного файла (рисунок 37).

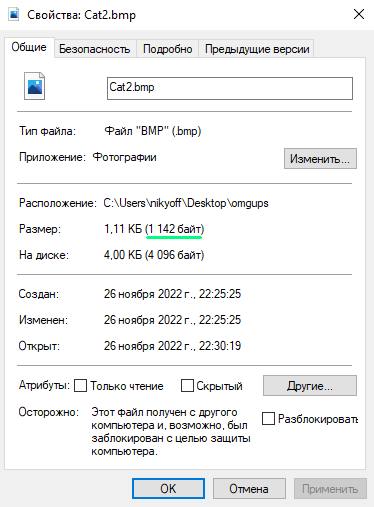


Рисунок  – Системный размер исходного файла

Системный размер исходного файла равен 1142 байтам, что в свою очередь соответствует полученному нами значению в поле «bfSize».

Зарезервированные поля не принимают участия при выполнении данного контрольного вопроса, поэтому перейдем к полю «bfOffBits». Для наглядности обратимся к рисунку 38.

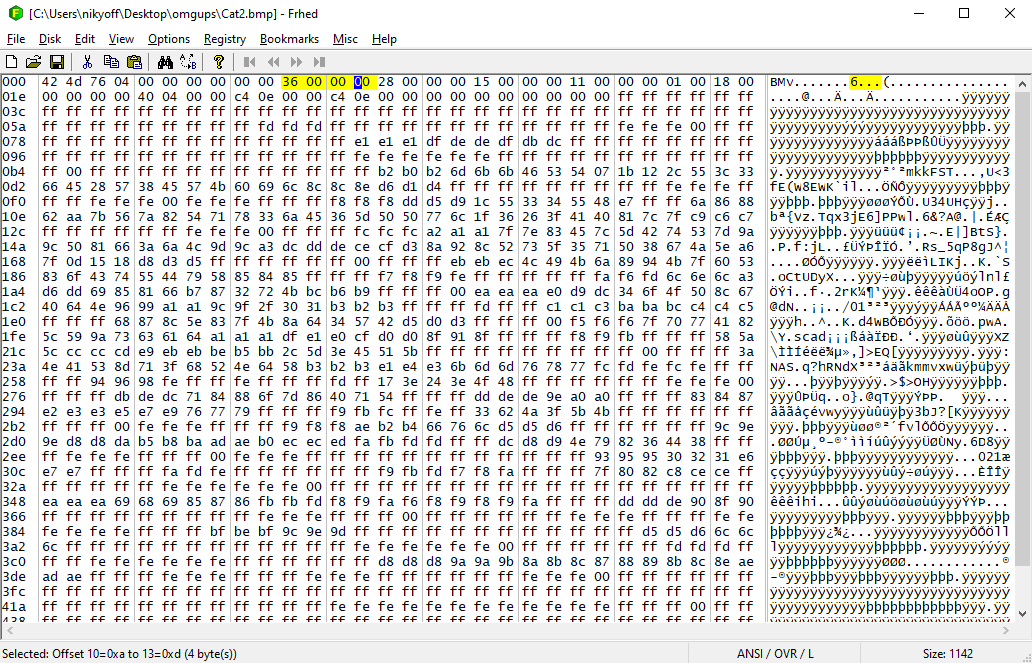


Рисунок  – «bfOffBits»

В результате получим последовательность байт «36 00 00 00», интерпретируем ее также в little-endian, то есть исходная последовательность байт равна «00 00 00 36», что равняется в десятичной системе счисления 54 (рисунок 39).

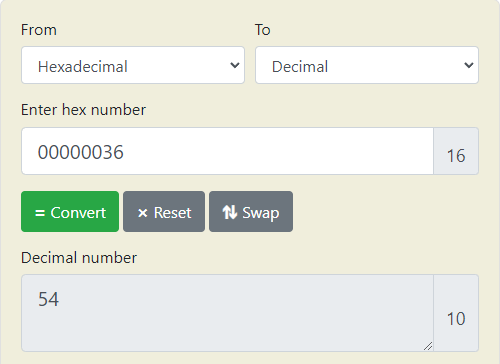


Рисунок  – Перевод систем счисления

Теперь перейдем к структуре «BITMAPINFO». Данная структура имеет зависимость от версии, поэтому для начала следует определить ее. Для определения версии требуется узнать значение первых четырех байт (DWORD), данное значение определяет как размер структуры, так и саму версию. Для наглядности обратимся к рисунку 40.

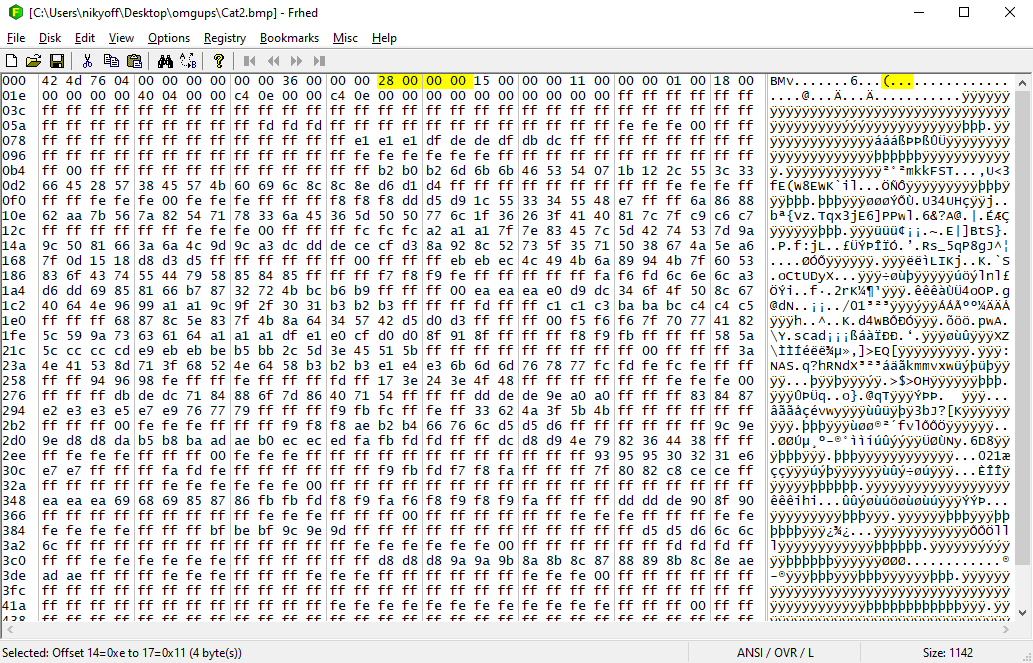


Рисунок  – Определение версии

В результате получаем следующую последовательность байт «28 00 00 00», ее следует интерпретировать в «little-endian». В результате исходная последовательность байт будет «00 00 00 28», переводя в десятичную систему счисления, получим 40 (рисунок 41).

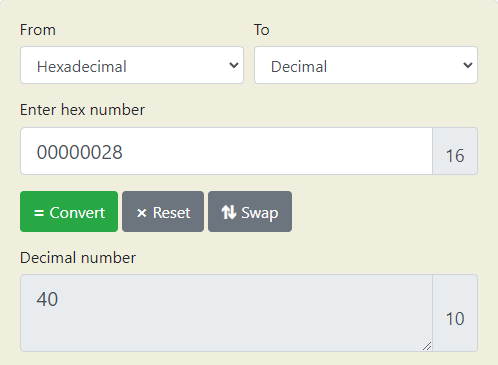


Рисунок  – Перевод систем счисления

В таблице 2 приведены соотношения версий и размера. Полученное нами значение соотносится с третьей версией. Исходя из версии, можно провести обзор структуры, для этого обратимся к таблице 3.

Для определения количества байт, занимаемых пиксельными данными, нам необходимо найти поля «biWidth», «biHeight» и «biBitCount».

Обратимся к рисункам 42 – 44 для того, чтобы наглядно показать соответствующие поля в frhed.

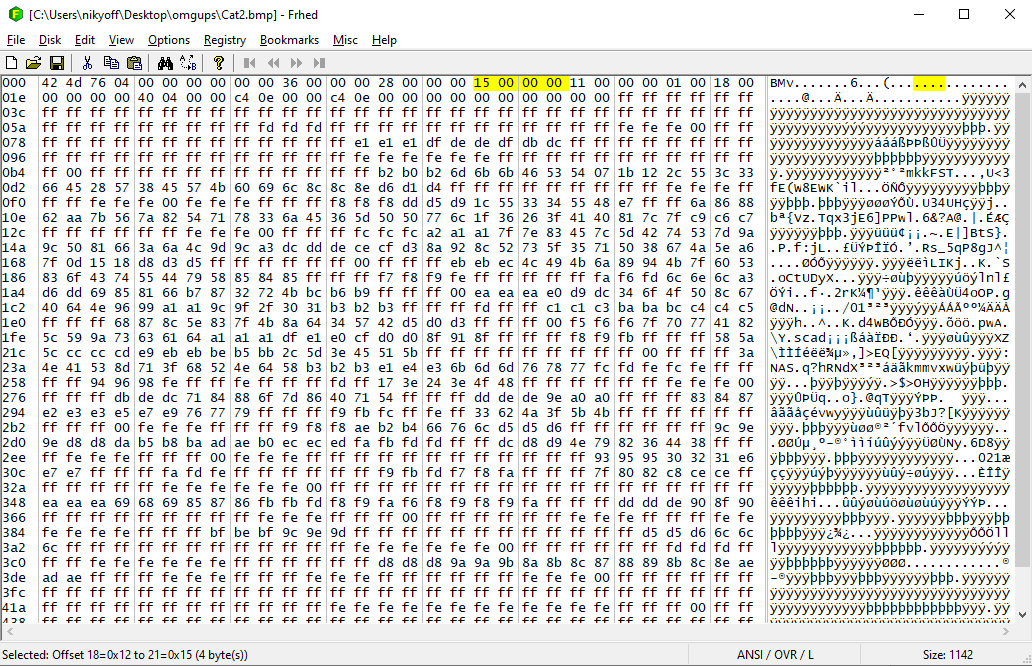


Рисунок  – «biWidth»

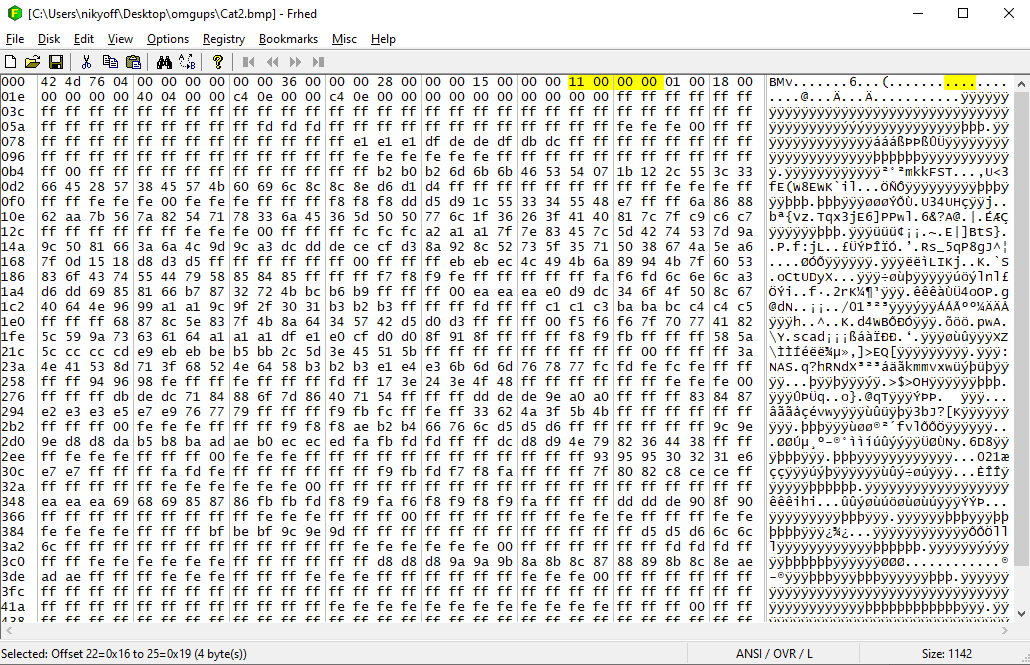


Рисунок  – «biHeight»

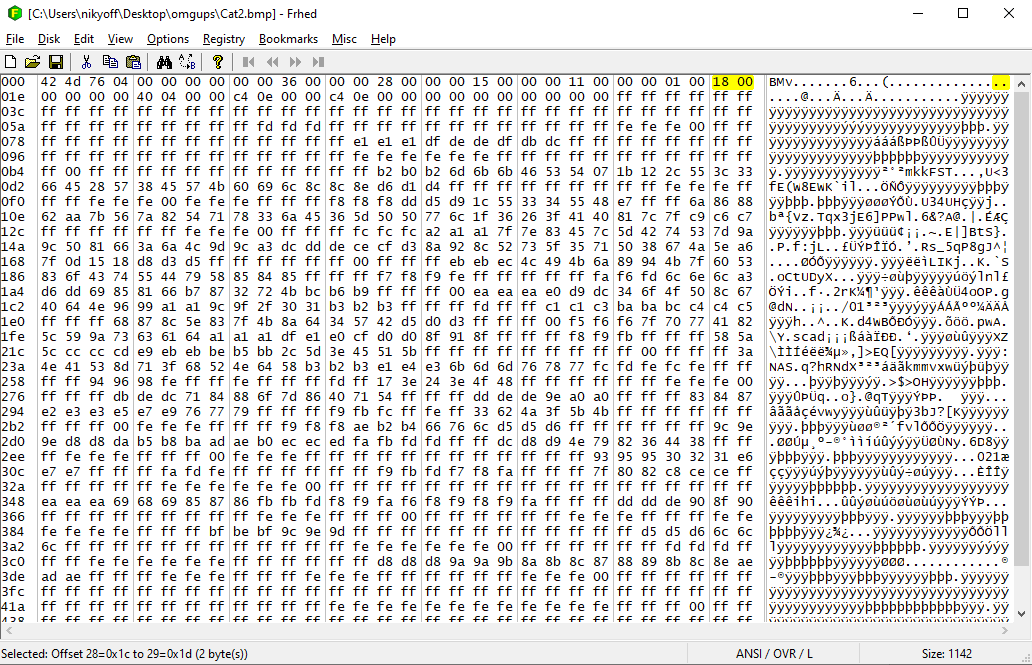


Рисунок  – «biBitCount»

Получаем следующие набор последовательностей байт:

biWidth: «15 00 00 00»;

biHeight: «11 00 00 00»;

biBitCount: «18 00».

Затем будем рассматривать их в little-endian последовательности:

biWidth: «00 00 00 15»;

biHeight: «00 00 00 11»;

biBitCount: «00 18».

Затем представим их в десятичной системе счисления (рисунки 45-47):

biWidth: 21;

biHeight: 17;

biBitCount: 24.

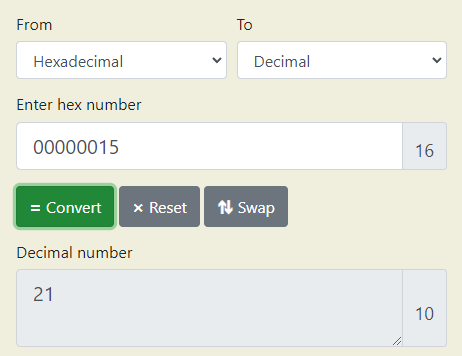


Рисунок  – Перевод систем счисления

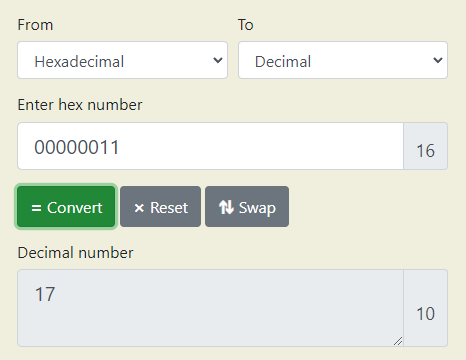


Рисунок  – Перевод систем счисления

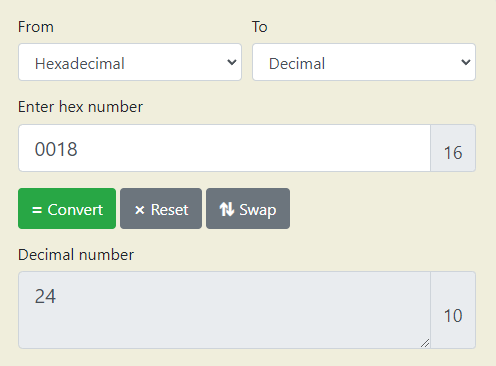


Рисунок  – Перевод систем счисления

Для нахождения количества байт, занимаемых пиксельными данными, необходимо перемножить три получившихся значения. В результате получим 8568 бит, что в байтовом эквиваленте соответствует числу 1071. Доказать истинность результата можно проведя небольшие вычисления. Структуры «BITMAPFILEHEADER» и «BITMAPINFO», в сумме занимают 54 байта, а общий размер файла равен 1142 байтам. Соответственно разница между этими значениями и образует количество байт, занимаемыми пиксельными данными. Но в результате вычисления, мы получаем отличное от ожидаемого значения, соответственно ранее вычисленная битность ставится под вопрос. Вероятнее всего, в данном случае поле «biSizeImage» не обнулено и будет указывать на настоящий размер пиксельных данных. Для его нахождения обратимся к рисунку 48.

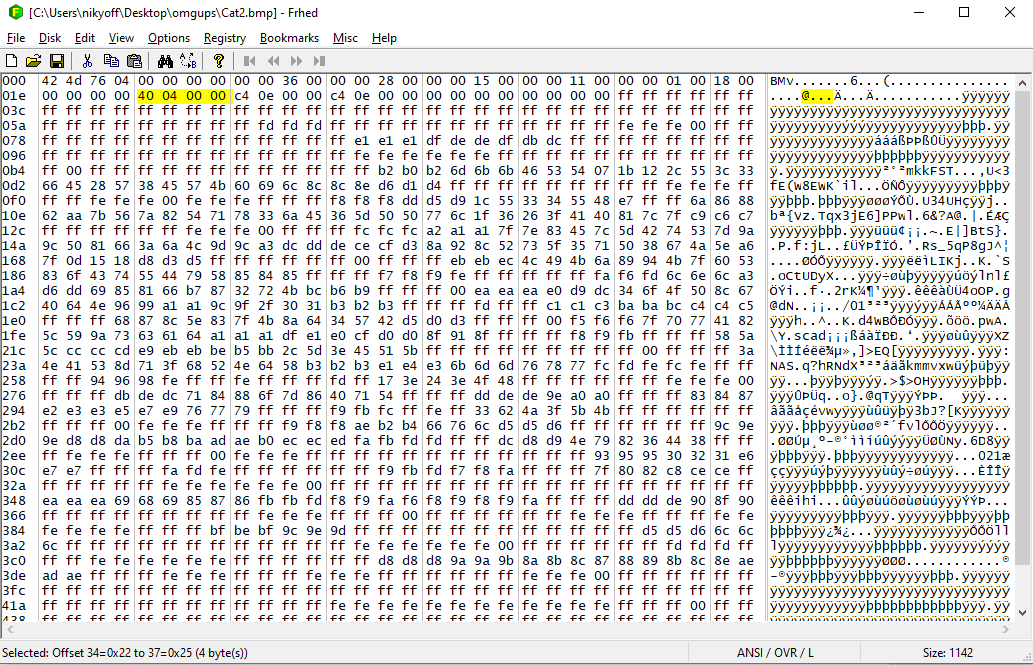


Рисунок  – «biSizeImage»

В результате получаем следующую последовательность байт «40 04 00 00», ее следует интерпретировать в «little-endian». В результате исходная последовательность байт будет «00 00 04 40», переводя в десятичную систему счисления, получим 1088 (рисунок 49).

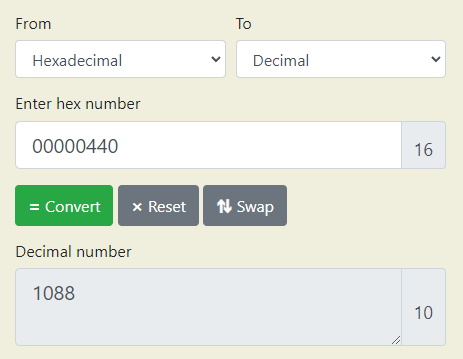


Рисунок  – Перевод систем счисления

Проводя повторную проверку полученных значений, действительно, разница между 1142 и 1088 дает размер суммы структур «BITMAPFILEHEADER» и «BITMAPINFO» тоесть 54 байта.

Откроем частичное представление файла, для этого необходимо выбрать начальное смещение и количество загружаемых байт. Необходимо загрузить 1088 байт с смещением в 54 байта (x36 в hex). Для наглядности обратимся к рисунку 50-51.

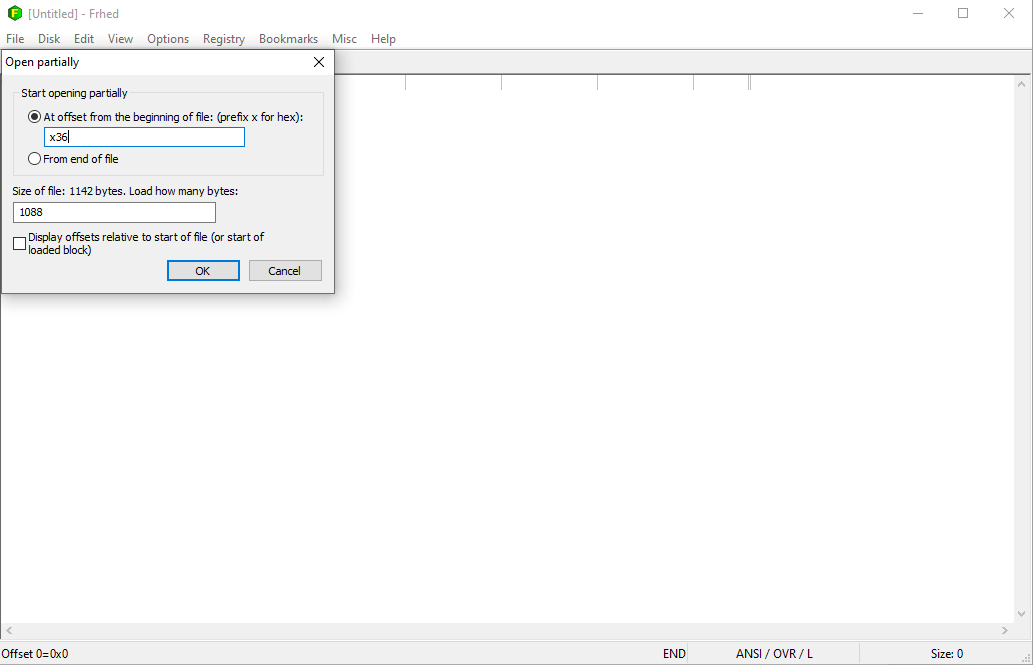


Рисунок  – Настройка частичного отображения со смещением

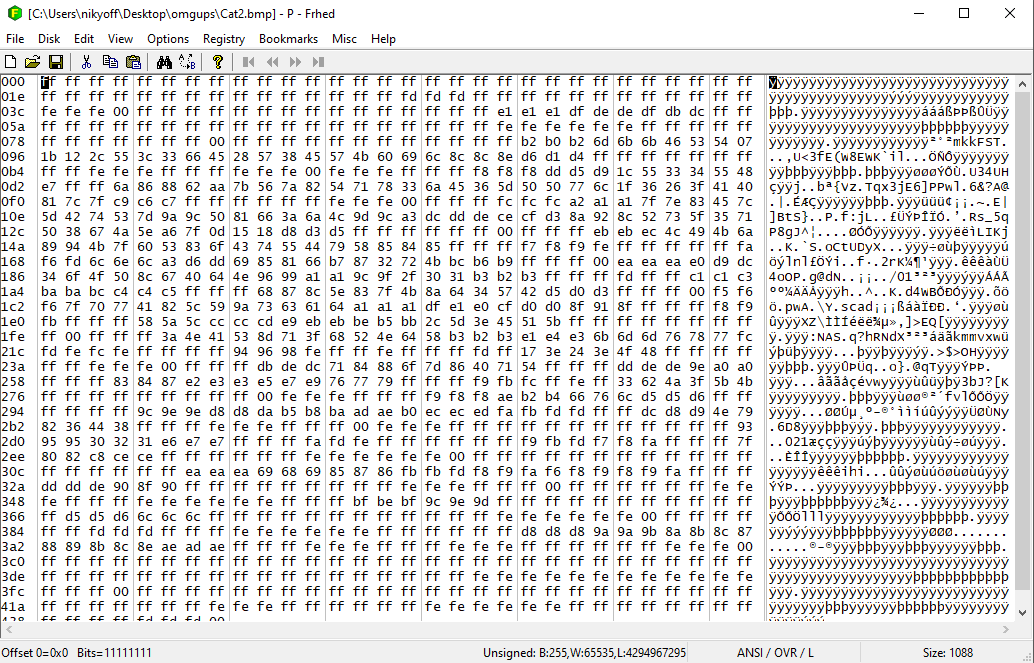


Рисунок  – Настройка частичного отображения со смещением

Исходя из ранее полученных значений, мы можем говорить о том, что каждый пиксель закодирован 3 байтами, что соответствует модели rgb, в свою очередь rgb представляет из себя смешивание трех базовых цветов – red (красный), green (зеленый), blue (синий) и кодирует каждый из базовых цветов в одном байте.

Пиксели образуют картинку, начиная от нижнего левого пикселя и двигаясь вправо и вверх. Таким образом, первые три байта отвечают за rgb модель представления первого левого нижнего пикселя. Следовательно при последовательном изменении значений первых трех байт на 00, то в результате получим черный цвет у первого пикселя (рисунок 52 – 54).

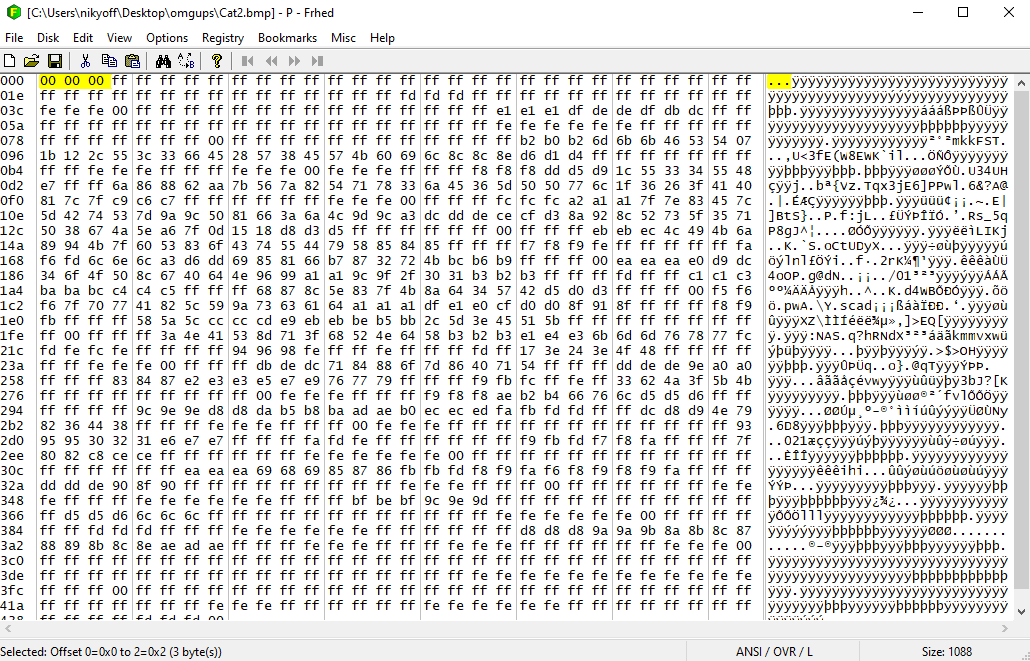


Рисунок  – Измененный набор байт первого пикселя



Рисунок  – Черный цвет пикселя



Рисунок  – Черный цвет пикселя в масштабе

Как и следовало ожидать, при изменении первых трех байт, изменяется цвет первого отображаемого пикселя на черный, так как «00 00 00» в rgb является черным цветом.

При изменении двух из трех байт на «ff», а третьего на «00», мы получим три различных результата:

1. «ff ff 00» – желтый цвет в результате смешивания красного и зеленого (рисунки 55 – 56)
2. «00 ff ff» – голубой цвет в результате смешивания зеленого и синего (рисунки 57 – 58)
3. «ff 00 ff» – лиловый цвет в результате смешивания красного и синего (рисунки 59 – 60)

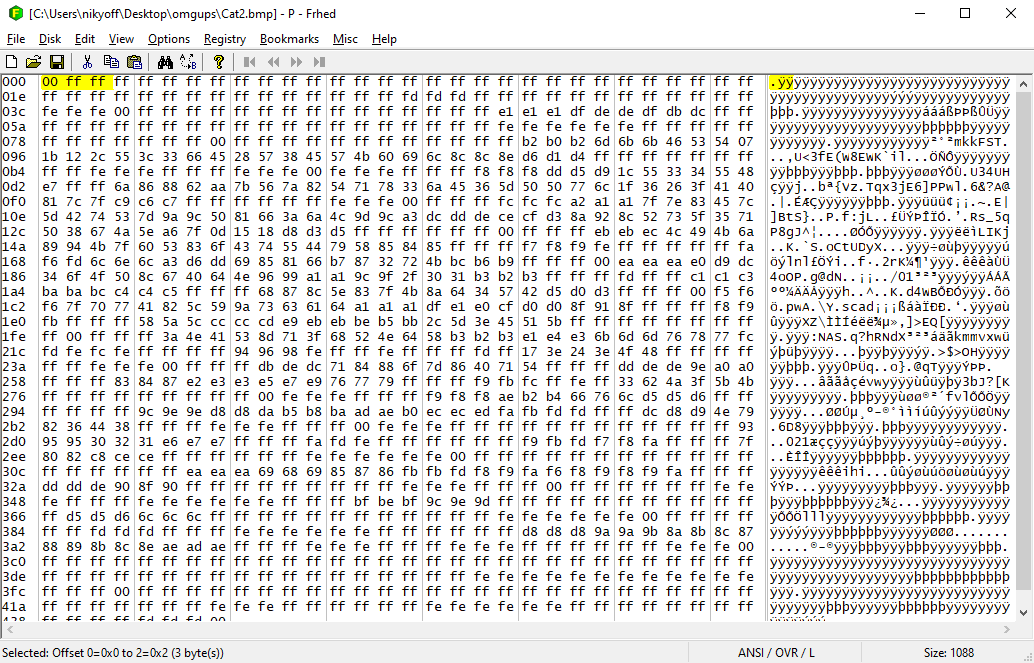


Рисунок  – «ff ff 00» в frhead



Рисунок  – Желтый цвет пикселя в масштабе

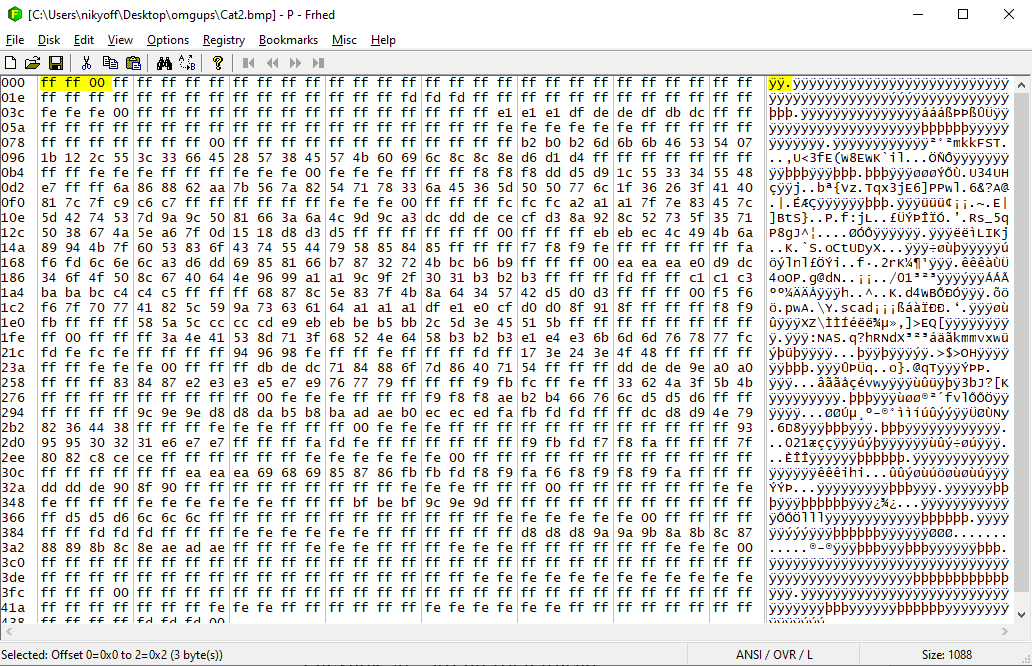


Рисунок  – «00 ff ff» в frhead



Рисунок  – Голубой цвет пикселя в масштабе

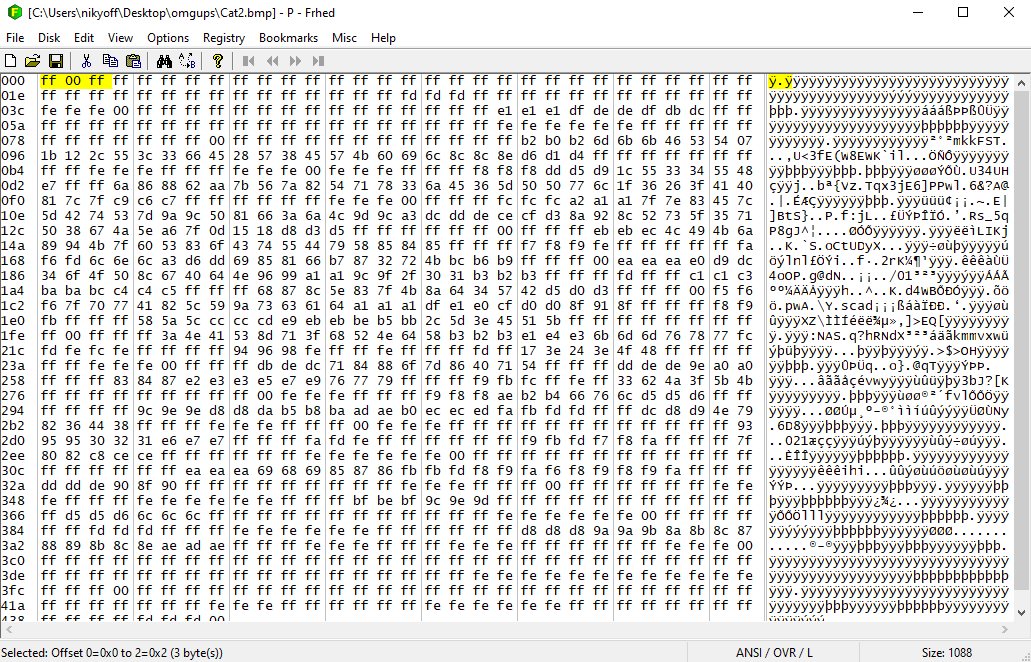


Рисунок  – Лиловый цвет пикселя в масштабе



Рисунок  – «ff 00 ff» в frhead

Вывод

В данной лабораторной работе, мы ознакомились с растровым форматом файла. Разобрали структуру bmp-формата, рассмотрели его байтовую структуру, а также ознакомились с процессами сжатия и алгоритмами растеризации линий.