BMP Image Files

Images are made up of small dots called "pixels". Images can be stored in a variety of ways. For this assignment, we focus on the BMP file format—and in particular, the 24-bit color format. In this format, each pixel consists of 3 bytes: red, green, and blue. This image data (pixels) is stored right after the header (54 bytes). The header is just a struct object (below) containing many details about the image, such as the width, height, and file format.

```
typedef struct {              // Total: 54 bytes
  uint16_t  type;             // Magic identifier: 0x4d42
  uint32_t  size;             // File size in bytes
  uint16_t  reserved1;        // Not used
  uint16_t  reserved2;        // Not used
  uint32_t  offset;           // Offset to image data in bytes from beginning of file (54 bytes)
  uint32_t  dib_header_size;  // DIB Header size in bytes (40 bytes)
  int32_t   width_px;         // Width of the image
  int32_t   height_px;        // Height of image
  uint16_t  num_planes;       // Number of color planes
  uint16_t  bits_per_pixel;   // Bits per pixel
  uint32_t  compression;      // Compression type
  uint32_t  image_size_bytes; // Image size in bytes
  int32_t   x_resolution_ppm; // Pixels per meter
  int32_t   y_resolution_ppm; // Pixels per meter
  uint32_t  num_colors;       // Number of colors
  uint32_t  important_colors; // Important colors
} BMPHeader;
```

After the header, the pixels are laid out in a row in the file, starting with the lower-left pixel. Each row must be a multiple of 4 bytes. To ensure that is the case, 0 to 3 bytes of padding may be added to the end of each row. This is a hex dump created with the xxd command.

```
$ xxd 6x6_24bit.bmp
0000000: 424d ae00 0000 0000 0000 3600 0000 2800  BM........6...(.
0000010: 0000 0600 0000 0600 0000 0100 1800 0000  ................
0000020: 0000 7800 0000 0000 0000 0000 0000 0000  ..x.............
0000030: 0000 0000 0000 0000 ff00 00ff ffff ffff  ................
0000040: ffff e7bf c8e7 bfc8 0000 0000 ff00 00ff  ................
0000050: ffff ffff ffff e7bf c8e7 bfc8 0000 00ff  ................
0000060: 0000 ff00 0000 0000 0000 ff00 80ff 0080  ................
0000070: 0000 00ff 0000 ff00 0000 0000 0000 ff00  ................
0000080: 80ff 0080 0000 0000 ff00 00ff ff80 00ff  ................
0000090: 8000 0080 ff00 80ff 0000 0000 ff00 00ff  ................
00000a0: ff80 00ff 8000 0080 ff00 80ff 0000       ..............
```

1. Circle the entire BMP header (the first 54 bytes).
2. Circle the file size and write the value here, in hex (_____) and decimal (_____).
   Note: In BMP, the bytes of integers are reversed. For example, if 167 (0x000000a7) is a uint32_t (4 bytes), it would be stored in the file as a7000000. This system of byte ordering is called "little endian". Yes, it's weird.
3. Circle the width and height.
4. Circle the pixel in the lower left corner of the image.
   a. Write the bytes in the order they appear in the file here, in hex: ___ ___ ___
   b. What color is it (in words)? _____
   c. What are the r, g, b values in decimal? r _____ g _____ b _____
5. Circle the entire lower row, including the padding. How many bytes is this row? _____ bytes
6. Circle the entire top row, including the padding. How many bytes is this row? _____ bytes

7. The following function should get the offset of a pixel at (x,y) within the image data (pixels).
   Example: To find the byte offset of the second pixel pixel from the left in the lower row of 6x6_24bit.bmp (represented in the hex dump above) you would use _get_offset_in_data(1, 5, 6, 6), which would return 6.

```
int _get_offset_in_data(int x, int y, int img_w, int img_h) {
    int padding_per_row =

    int row_start_idx =

    int idx_in_row =

    return

}
```

8. Fill in the following type and function:

```
typedef struct { unsigned char r, g, b; } _Color;

_Color _get_pixel(BMPImage* image, int x, int y) {




}
```

```
typedef struct {
    BMPHeader header;
    unsigned char* data;
} BMPImage;

typedef struct {              // Total: 54 bytes
  uint16_t  type;             // Magic identifier: 0x4d42
  uint32_t  size;             // File size in bytes
  uint16_t  reserved1;        // Not used
  uint16_t  reserved2;        // Not used
  uint32_t  offset;           // Offset to image data in bytes from beginning of file (54
bytes)
  uint32_t  dib_header_size;  // DIB Header size in bytes (40 bytes)
  int32_t   width_px;         // Width of the image
  int32_t   height_px;        // Height of image
  uint16_t  num_planes;       // Number of color planes
  uint16_t  bits_per_pixel;   // Bits per pixel
  uint32_t  compression;      // Compression type
  uint32_t  image_size_bytes; // Image size in bytes
  int32_t   x_resolution_ppm; // Pixels per meter
  int32_t   y_resolution_ppm; // Pixels per meter
  uint32_t  num_colors;       // Number of colors
  uint32_t  important_colors; // Important colors
} BMPHeader;
```