# Formato Bitmap

Canoi Gomes

23 de Junho de 2024

## Conteúdo

1	dução	3				
2	Cabe	eçalho	3			
	2.1	Cabeçalho de Arquivo	4			
	2.2	Cabeçalho de Informação	5			
3	3 Criando um Bitmap em C					
4	Lend	o e Exibindo um Bitmap	11			

#### 1 Introdução

No texto anterior eu dei uma rápida explicação sobre como funciona um arquivo binário, como construir o seu próprio e fazer um programa para escrever e ler o mesmo.

O formato de arquivo bitmap (possui a extensão .bmp) é um formato normalmente utilizado para guardar os dados de pixel de uma image. É um dos formatos que são conhecidos por geralmente não ter compressão, então diferente de um .png, onde existe um algoritmo de compressão para codificar e decodificar, resultando em um arquivo com tamanho menor, o .bmp não se preocupa tanto com tamanho e preserva a facilidade de manipular o arquivo.

#### 2 Cabeçalho

O formato bitmap possui dois cabeçalhos, um cabeçalho de arquivo e um cabeçalho de informação. O de arquivo tem 14 bytes, e o de informação, bom, aqui pode variar bastante, o que estarei usando é um de 40 bytes de tamanho, chamado BITMAPINFOHEADER. Existem outros, como um bem mais simples de 12 bytes chamado BITMAPCOREHEADER que você pode dar uma olhada se quiser algo mais simples de implementar.

Name		Size	Description	
ader		14 bytes	Windows Structure: BITMAPFILEHEADER	
Signature		2 bytes	'BM'	
FileSize		4 bytes	File size in bytes	
reserved		4 bytes	unused (=0)	
DataOffse	t	4 bytes	File offset to Raster Data	
Header		40 bytes	Windows Structure: BITMAPINFOHEADER	
Size		4 bytes	Size of InfoHeader =40	
Width		4 bytes	Bitmap Width	
Height		4 bytes	Bitmap Height	
Planes		2 bytes	Number of Planes (=1)	
BitCount		2 bytes	Bits per Pixel  1 = monochrome palette. NumColors = 1  4 = 4bit palletized. NumColors = 16  8 = 8bit palletized. NumColors = 256  16 = 16bit RGB. NumColors = 65536 (?)  24 = 24bit RGB. NumColors = 16M	
Compression ImageSize XpixelsPerM		4 bytes	Type of Compression  0 = BI_RGB no compression  1 = BI_RLE8 8bit RLE encoding  2 = BI_RLE4 4bit RLE encoding  (compressed) Size of Image  It is valid to set this =0 if Compression = 0  horizontal resolution: Pixels/meter	
		4 bytes		
		4 bytes		
YpixelsPer	pixelsPerM 4 bytes		vertical resolution: Pixels/meter	
ColorsUse	ColorsUsed 4 bytes		Number of actually used colors	
ColorsImp	ColorsImportant 4 bytes		Number of important colors 0 = all	
lorTable		4 * NumColors bytes	present only if Info.BitsPerPixe1 <= 8 colors should be ordered by importance	
F	Red	1 byte	Red intensity	
	Freen	1 byte	Green intensity	
	Blue .	1 byte	Blue intensity	
	eserved	1 byte	unused (=0)	
repeated NumColors times				
ster Data		Info.ImageSize bytes	The pixel data	

## 2.1 Cabeçalho de Arquivo

O cabeçalho de arquivo possui 14 bytes de tamanho, como descrito na imagem acima.

Campo	Bytes	Descrição
Validação	2 bytes	É usado para a validação do formato, seria seu
		magic number, geralmente é preenchido com
		"BM".
Tamanho	4 bytes	Tamanho do arquivo em bytes.
Reservado	4 bytes	Geralmente não é utilizado.
Offset	4 bytes	A partir de onde podemos começar a ler os dados
		dos pixels do arquivo. Em uma imagem padrão,
		que não tenha informações extra antes dos pixels,
		esse valor é 54 geralmente, que seria a soma dos
		valores de ambos os cabeçalhos (14 + 40).

Poderia representá-lo assim em C.

```
struct FileHeader {
   char magic[2];
   char size[4];
   char reserved[4];
   char offset[4];
}
```

Listing 1: Cabeçalho de Arquivo em C

## 2.2 Cabeçalho de Informação

O cabeçalho de informação tem 40 bytes de tamanho, e é nele que geralmente vão as informações relativas a imagem como sua paleta de cores, a quantidade de bits que estou usando para representar um pixel, as dimensões, entre outras coisas.

Campo	Bytes	Descrição	
Tamanho do Ca-	4 bytes	Aqui irá conter o tamanho do cabeçalho em bytes,	
beçalho		geralmente esse valor será 40 mesmo.	
Largura	4 bytes	Largura da imagem	
Altura	4 bytes	Altura da imagem	
Planos	2 bytes	Número de planos (também não sei para que	
		serve, geralmente é 1)	
Bits Por Pixel	2 bytes	Esse campo é interessante, nele eu especifico a	
		quantidade de bits que quero usar para represen-	
		tar um pixel. É comum esse valor ser 24 (RGB	
		com 3 bytes, ou seja 1 byte para cada canal) ou	
		32 (RGBA com 4 bytes). Porém também pode	
		ser 16 (também é RGB, mas é um RGB565, 5 bits	
		para R, 6 para G e 5 para B). Se setarmos valores	
		abaixo de 16, como 8 ou 4, precisamos especifi-	
		car uma paleta de cores logo depois do cabeça-	
		lho. 8 bits por exemplo, preciso de uma paleta	
		com 256 cores, 4 bits uma paleta com 16 cores e	
		assim vai.	
Compressão	4 bytes	Aqui basicamente você pode setar um modo de	
		compressão, mas como não estamos utilizando	
		nenhum geralmente esse valor será zero mesmo.	
Tamanho da	4 bytes	Aqui vai representar o tamanho da imagem após	
Imagem		a compressão. Se não tiver nenhuma compres-	
		são, como é o nosso caso, esse valor pode ser zero	
		também.	

XpixelsPerM	4 bytes	
YpixelsPerM	4 bytes	
Cores Usadas	4 bytes	
Cores Importan-	4 bytes	
tes		

Posso representar esse cabeçalho assim em C:

```
struct InfoHeader {
    char header_size[4];
    char width[4];
    char height[4];
    char planes[2];
    char bits_per_pixel[2];
    char compression[4];
    char image_size[4];
    char x_pixels[4];
    char y_pixels[4];
    char colors_used[4];
    char colors_important[4];
};
```

Listing 2: Cabeçalho de Informação em C

## 3 Criando um Bitmap em C

Agora vamos botar um pouco a mão na massa, vamos escrever um programinha em em C que fazendo uso dos cabeçalhos consiga fazer uma imagem bitmap que possamos visualizar em qualquer editor/visualizador de imagens.

```
#include <stdio.h>
struct __attribute__((__packed__)) FileHeader {
     short magic;
     int size;
     int reserved;
     int offset;
8 };
struct __attribute__((__packed__)) InfoHeader {
     int size;
     int width;
     int height;
     short planes;
     short bits_per_pixel;
     int compression;
     int image_size;
     int x_pixels;
     int y_pixels;
     int colors_used;
     int colors_important;
22 };
```

Listing 3: Cabeçalhos

Aqui estou me aproveitando alguns recursos do compilador, o

\_\_attribute\_\_((\_\_packed\_\_)) serve para ele tentar comprimir meu formato de maneira a evitar bytes vazios entre os campos. Se eu não tivesse feito isso, entre os campos de **magic** e **size** do meu FileHeader, iriam existir dois bytes extras, então no fim meu struct teria 16 bytes de tamanho. Mas como nesse caso cada byte faz diferença, eu pedi para o compilador evitar fazer isso pois preciso preservar os 14 bytes. Fique atento pois dependendo do compilador que você estiver usando pode

ser que isso não funcione.

Agora vamos usar isso para criar nossa imagem. Vou criar uma imagem pequena para facilitar a explicação, terá dimensões 2x2 e com formato RGB. Isso quer dizer que nossa imagem terá no total 4 pixels, e se cada pixel tem 3 bytes de tamanho, então no fim terei 12 bytes de dados. Porém o formato bitmap exige que você preencha cada linha com 1 byte extra para cada pixel para manter o alinhamento de 4 bytes para cada um. Outro detalhe importante é que os pixels quando escritos no documento são escritos ao contrário, então fica BGR, no fim terei algo assim: BGR BGR 00. Considerando os bytes extras, meus dados terão 16 bytes de tamanho, somando isso ao cabeçalho, terei um arquivo de 70 bytes de tamanho.

Vamos tentar escrever isso.

```
int main(int argc, char** argv) {
      FileHeader fh;
      InfoHeader ih;
      fh.magic = 0x4D42; // M(4D) B(42)
      fh.size = 70; // 70 bytes
      fh.offset = 54;
      ih.size = 40;
10
      ih.width = 2;
      ih.height = 2;
12
      ih.planes = 1;
13
      ih.bits_per_pixel = 24;
      ih.compression = 0;
15
      ih.image_size = 0;
      ih.x_pixels = 0;
17
      ih.y_pixels = 0;
18
      ih.colors_used = 0;
```

```
ih.colors_important = 0;
      FILE* fp = fopen("image.bmp", "wb");
22
23
      fwrite(&fh, 1, sizeof(fh), fp);
     fwrite(&ih, 1, sizeof(ih), fp);
      char data[16] = {
      // |b
             lg lr
                        ||b |g |r |extra bytes
         0x00, 0x00, 0xff, 0x00, 0xff, 0x00, 0x00, 0x00,
         0xff, 0x00, 0x00, 0xff, 0xff, 0xff, 0x00, 0x00,
     };
     fwrite(data, 1, sizeof(data), fp);
33
      fclose(fp);
      return 0;
37 }
```

Listing 4: Escrevendo um Bitmap

E no fim, essa é a imagem que nós acabamos de criar:

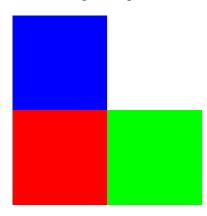


Figura 1: Imagem Gerada

### 4 Lendo e Exibindo um Bitmap

Para esse exemplo irei utilizar a biblioteca SDL2, que é uma framework que me permite criar uma janela e desenhar nela, fora input, áudio e outras funcionalidades. Recomendo você dar uma olhada com calma nela depois, eu gosto dela principalmente pelo suporte multiplataforma.