# ［演習 03-04］　ビットマップファイル

ビットマップファイル（BMP ファイル，bitmap ファイル）は，標準的な画像ファイルの 1 つである．今後，本演習では，BMP ファイルを用いて，画像情報処理プログラムを作成する．

本資料は，ビットマップファイルのフォーマットについて説明するものである．

## 1.　構成

BMP ファイルには，windows 版と呼ばれるものと OS/2 版と呼ばれるものとがあるが，本演習では，windows 版と呼ばれるファイルを用いて，画像情報処理プログラムを作成することとする．

windows 版の BMP ファイルの典型的な構成を表 3.1 に示す．表 3.1 に示されるように，BMP ファイルは，画像サイズや色数などの画像に関する情報を示すデータ（このような付加的なデータのことを一般にヘッダと呼ぶ）と，その後に続く各画素の値を示すデータから構成されている．以下，前者をヘッダ部，後者を画像データ部と呼ぶことにする．

BMP ファイルは，ヘッダ部に格納された値によって，その後に続くデータの構成が少し異なってくる．しかしながら，本演習では，表 3.1 に示されるような典型的な構成の BMP ファイルを使用することとする．

表 3.1

（補足）表においては，n バイト目の値を d[n]と表している．また，“0x”は，16 進数であることを示している．たとえば，0x42 は，16 進数の 42 であり，2 進数で示せば 0100　0010，10 進数で示せば 66 である．

| | バイト番号 | 内容 |
|---|---|---|
| ヘッダ部 | 0 | ファイルタイプ．d[0]は，文字 B を表す ASCII コード 0x42，d[1]は，文字 M を表す ASCII コード 0x4d． |
| | 1 | |
| | 2 | ファイルサイズ（バイト）． |
| | 3 | d[5] d[4] d[3] d[2]と順番に並べた 32 ビットでファイルサイズを表す． |
| | 4 | （例）d[5]=0x00，d[4]=0x03，d[3]=0xDE，d[2]=0x38 ならば，ファイルサイズは 0x0003DE38 であり，10 進数で示せば 253,496 バイトである． |
| | 5 | |
| | 6 | 予約領域． |
| | 7 | d[6]=0x00，d[7]=0x00． |
| | 8 | 予約領域． |
| | 9 | d[8]=0x00，d[9]=0x00． |
| | 10 | 画像データの先頭までのオフセット（バイト）．d[13] d[12] d[11] d[10]と順番に並べた 32 ビットでオフセットを表す．演習で使用するフォーマットでは，d[13]=0x00，d[12]=0x00，d[11]=0x00，d[10]=0x36，すなわち，オフセットは 0x00000036 であり，10 進数で示せば 54 バイトである． |
| | 11 | |
| | 12 | |
| | 13 | |

求め方の詳細については
後述する補足に記載

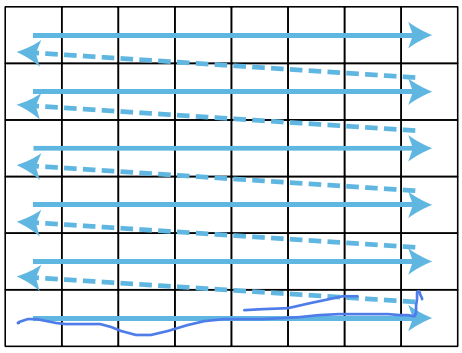
| | | |
|---|---|---|
| | 14 | 情報ヘッダサイズ（バイト）．d[17] d[16] d[15] d[14]と順番に並べた |
| | 15 | 32 ビットで情報ヘッダサイズを表す．演習で使用するフォーマットでは， |
| | 16 | d[17]=0x00，d[16]=0x00，d[15]=0x00，d[14]=0x28，すなわち，情 |
| | 17 | 報ヘッダサイズは 0x00000028 であり，10 進数で示せば 40 バイトである． |
| | 18 | 画像の幅（ピクセル）．d[21] d[20] d[19] d[18]と順番に並べた 32 ビ |
| | 19 | ットで画像の幅を表す． |
| | 20 | （例）d[21]=0x00，d[20]=0x00，d[19]=0x01，d[18]=0x60 ならば， |
| | 21 | 画像の幅は 0x00000160 であり，10 進数で示せば 352 ピクセルである． |
| | 22 | 画像の高さ（ピクセル）．d[25] d[24] d[23] d[22]と順番に並べた 32 |
| | 23 | ビットで画像の高さを表す． |
| | 24 | （例）d[25]=0x00，d[24]=0x00，d[23]=0x00，d[22]=0xF0 ならば， |
| | 25 | 画像の高さは 0x000000F0 であり，10 進数で示せば 240 ピクセルである． |
| | 26 | 色プレーン数．d[27] d[26]と順番に並べた 16 ビットで色プレーン数を表 |
| | 27 | す．常に，d[27]=0x00，d[26]=0x01，10 進数で示せば 1 である． |
| | 28 | 1 画素当たりのビット数．d[29] d[28]と順番に並べた 16 ビットで表す． |
| | 29 | 演習では，d[29]=0x00，d[28]=0x18，10 進数で示せば 24 ビットである． |
| | 30 | 圧縮方式． |
| | 31 | d[33] d[32] d[31] d[30]と順番に並べた 32 ビットで圧縮方式を表す． |
| | 32 | 演習では，d[33]=0x00，d[32]=0x00，d[31]=0x00，d[30]=0x00，10 |
| | 33 | 進数で示せば 0 であり，非圧縮方式である． |
| | 34 | 画像データサイズ．d[37] d[36] d[35] d[34]と順番に並べた 32 ビット |
| | 35 | で画像データサイズを表す． |
| | 36 | 圧縮方式が非圧縮の場合には，d[37]=0x00，d[36]=0x00，d[35]=0x00， |
| | 37 | d[34]=0x00 とする． |
| | 38 | 水平解像度（ピクセル/メートル）．d[41] d[40] d[39] d[38]と順番に |
| | 39 | 並べた 32 ビットで水平解像度を表す． |
| | 40 | （例）d[41]=0x00，d[40]=0x00，d[39]=0x0B，d[38]=0x12 ならば， |
| | 41 | 水平解像度は 0x00000B12 であり，10 進数で示せば 2834 ピクセル／メートル，72 ピクセル／インチ程度である． |
| | 42 | 垂直解像度（ピクセル/メートル）．d[45] d[44] d[43] d[42]と順番に |
| | 43 | 並べた 32 ビットで垂直解像度を表す． |
| | 44 | （例）d[45]=0x00，d[44]=0x00，d[43]=0x0B，d[42]=0x12 ならば， |
| | 45 | 水平解像度は 0x00000B12 であり，10 進数で示せば 2834 ピクセル／メートル，72 ピクセル／インチ程度である． |
| | 46 | パレットの色数．d[49] d[48] d[47] d[46]と順番に並べた 32 ビットで |
| | 47 | パレットの色数を表す．演習では，d[47]=0x00，d[46]=0x00， |
| | 48 | d[45]=0x00，d[44]=0x00 であり，10 進数で示せば 0 である． |
| | 49 | |

| | | |
|---|---|---|
| | 50 | 重要な色数．d[53] d[52] d[51] d[50]と順番に並べた 32 ビットで重要な色数を表す．演習では，d[53]=0x00，d[52]=0x00，d[51]=0x00，d[50]=0x00，10 進数で示せば 0 であり，すべての色が重要であることを示す． |
| | 51 | |
| | 52 | |
| | 53 | |
| 画像データ部 | 54 | 画面の 1 番下のラインの左端の画素の値（B 信号の値） |
| | 55 | 画面の 1 番下のラインの左端の画素の値（G 信号の値） |
| | 56 | 画面の 1 番下のラインの左端の画素の値（R 信号の値） |
| | 57 | 以下，図 3.1 に示すような画素の順番に，B 信号，G 信号，R 信号の値が繰り返し格納される． |
| | … | |

## 2. 画像データの格納順序

画面における画素の位置と，BMP ファイルの画像データ部に格納される画素データの順番との関係を図 3.1 に示す．図に示されるように，BMP ファイルでは，画面の左下から右上に向かう順番で画素の値がファイルに格納される．

また，演習で使用するような，各画素が R 信号（Red）8 ビット，G 信号（Green）8 ビット，B 信号（blue）8 ビットの計 24 ビットで構成される BMP ファイルの場合には，各画素に対して，B 信号，G 信号，R 信号の順番でデータがファイルに格納される．



左図に示す順番で，画素の値がファイルに格納される．
その際，各画素においては，
B 信号，G 信号，R 信号の順番でファイルに格納される．
（表 3.1 のバイト番号 54，55，56 の内容と照らし合わせてみよ．）

図 3.1

## 3. 4 バイトアラインメント

### 3.1 ライン単位のアラインメント

BMP ファイルでは，画素の値は 1 ライン単位（画面上，横並びの 1 行単位）でファイルに格納されることになっており，その際，1 ライン当たりのバイト数は 4 の倍数バイトであることが定められている．そのため，1 ライン当たりのデータ数が 4 の倍数でない場合には，各ラインのデータの最後に 0 を追加して，4 の倍数バイトにする必要がある．

本演習では，画素幅が 4 の倍数であるような画像のみを扱うことにする．これにより，1 ライン当たりのバイト数は必ず 4 の倍数になるため，各ラインにおけるデータの挿入が不要になる．（よって，プログラム作成の際には，1 ラインごとのバイト数の調整については考えなくてよい．）

### 3.2 ファイル単位のアラインメント

BMP ファイルでは，ファイルサイズが 4 の倍数バイトであることが定められている．そのため，ファイルサイズが 4 の倍数バイトでない場合には，ファイルの最後に 0 を追加して，4 の倍数バイトにする必要がある．

たとえば，

ヘッダ部のサイズ　　：54 バイト
画像のサイズ　　：352×240
1 画素当たりのビット数　　：24 ビット（3 バイト）

ならば，実質的なバイト数は，

54+352×240×3=253494 バイト

であるが，253494 を 4 で割ると，商が 63373 であり，余りが 2 である．すなわち，

253494=4×63373+2

であるため，ファイルの最後に 0 を 2 バイト追加して，

ファイルサイズ　　：253496 バイト

にする必要がある．

**［補足］4 バイトで表された 10 進数値について**

図 3.2 は，4 バイトのデータを順に並べたときの 10 進数値の求め方を示した概念図である．ここでは例として，上位から順に，0x01，0x23，0xCD，0xEF の 4 バイトを並べた場合について示してある．

この例において，今求めたいことは「上記 4 バイトを並べた 0x0123CDEF は，どのような 10 進数を表しているのか」ということである．これは，以下の手順で求めることができる．

00000001,00100011,11001101,11101111 という 2 進数値を持つ 4 つの変数があるとき，

(1) 00000001001000111100110111101111 という 2 進数値を持つ 1 つの変数を作る．

(2) その変数が表す 10 進数値を示す．

（上記 2 進数値は，それぞれ 8 ビット，32 ビットで表されているが，ビット幅は，特に意味を持たない．すなわち，先に示された 2 進数の上位に 0 が連なっていても特に問題はない．）

ここで，上記 (1) は，以下の 4 つの値を加算することによって得ることができる．

```
00000001000000000000000000000000
00000000001000110000000000000000
00000000000000001100110100000000
00000000000000000000000011101111
```

（言うまでもなく，これらの加算結果は，00000001001000111100110111101111 である．）

図 3.2 は，上記加算を“効率的に”実現するための手順を示した概念図である．図に示された手順により，前述した処理と同じ結果を得ることができる．手順を順に追って，処理の内容をよく理解しておこう．

なお，図 3.2 の最下方に参考として記載されているように，上記方法で得られる結果は，ビットシフト演算を使用しなくとも得ることができる．これは，以下のような考え方に基づいている．

- 2 進数を 1 ビット左にシフトすることは，10 進数で考えれば「× 2」することに相当する．
- 2 進数を 8 ビット左にシフトすることは，「× 2」を 8 回行うことに相当するので，10 進数で考えれば「$\times 2^8$」（すなわち，「$\times 256^1$」）することに相当する．
- 同様に，16 ビット左シフトは「$\times 2^{16}$」（すなわち，「$\times 256^2$」），24 ビット左シフトは「$\times 2^{24}$」（すなわち，「$\times 256^3$」）することに相当する．

上記の意味する所もよく理解しておこう．

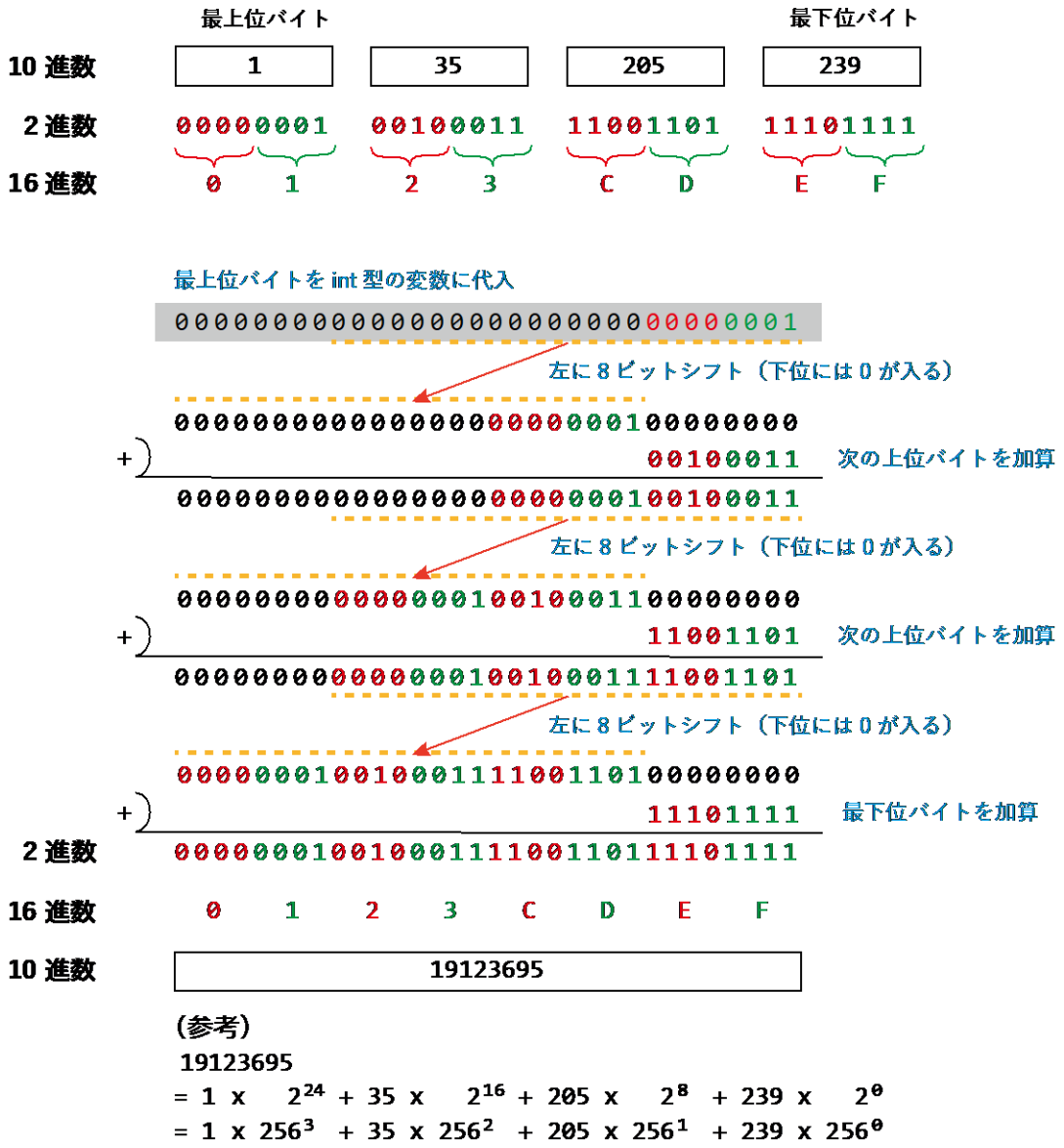

最上位バイト
最下位バイト
10 進数
1
35
205
239
2 進数
00000001
00100011
11001101
11101111
16 進数
0 1 2 3 C D E F
最上位バイトを int 型の変数に代入
00000000000000000000000000000001
左に 8 ビットシフト（下位には 0 が入る）
00000000000000000000000100000000
+)
00100011
次の上位バイトを加算
00000000000000000000000100100011
左に 8 ビットシフト（下位には 0 が入る）
00000000000000000001001000110000000
+)
11001101
次の上位バイトを加算
00000000000000000100100011100001101
左に 8 ビットシフト（下位には 0 が入る）
00000001001000111100110100000000
+)
11101111
最下位バイトを加算
2 進数
00000001001000111100110111101111
16 進数
0 1 2 3 C D E F
10 進数
19123695
(参考)
19123695
= 1 x 2^24 + 35 x 2^16 + 205 x 2^8 + 239 x 2^0
= 1 x 256^3 + 35 x 256^2 + 205 x 256^1 + 239 x 256^0

図 3.2

図 3.2 に示された手順を参考にして作成したプログラムの例を ex_03_ref_1 に，その実行結果を ex_03_ref_1_result に示す．プログラムが何を行っているのか，実行結果を参考に，コードを 1 行ずつ読み進め，処理の内容をよく理解しておこう．

なお，ex_03_ref_1 においては，今後の演習の都合上，

- 配列 ary をグローバル変数として宣言
- 関数 cal_value を定義し，これを用いて，4 バイトで表される 10 進数値を算出

している．ここで，関数 cal_value は，配列 ary の添え字 lsb_index から始まる連続する byts バイトを逆順に並べたときに表される 10 進数値を求める関数である．たとえば，cal_value(0, 4)は，配列 ary[0]から始まる連続する 4 バイトである ary[0]，ary[1]，ary[2]，ary[3]を，逆順に並べたとき（すなわち，ary[3]，ary[2]，ary[1]，ary[0]という順番で並べたとき）の 10 進数値を返す．

プログラム中に記載された << は，ビット単位のシフト演算子と呼ばれるものである．たとえば，value << 8 は，変数 value の値を 8 ビット左にシフトした値を求めるものである．ここで，上記演算では，空いた下位の 8 ビットには，0 が挿入されることが定められている．

また，<<= は複合代入演算子と呼ばれるものである．たとえば，value <<= 8 は，value を 8 ビットシフトするということを意味しており，value = value << 8 と等価である．

ex_03_ref_1

```
/*--- ex_03_ref_1 ---*/

#include <stdio.h>

int cal_value(int msb_index, int bytes);

unsigned char ary[4] = { 239, 205, 35, 1 };

int main(void)
{
    int i;
    int value;

    //--- 配列内容の確認 ---
    printf("＜配列＞\n");
    for (i = 0; i < 4; i++)
        printf("ary[%d]：（10進数）%3d,（16進数）%02X\n", i, ary[i], ary[i]);
    //// %d   : 10進数で表示する
    //// %3d  : 3桁の幅を確保して，10進数で表示する
    //// %X   : 16進数で表記する
    //// %2X  : 2桁の幅を確保して，16進数で表示する
    //// %02X : 2桁の幅を確保して，16進数で表示する（上位の桁がない場合には 0 を表示）

    //--- 4バイトで表される10進数値を求める ---
    value = cal_value(0, 4);

    //--- 結果表示 ---
    printf("\n＜結果＞\n");
    printf("（10進数）%d，（16進数）%08X\n", value, value);

    return 0;
}

int cal_value(int lsb_index, int bytes) // LSB : Least Significant Byte（最下位バイト）
{
    int i;
    int value = ary[lsb_index + bytes - 1];

    printf("\n＜途中経過＞\n");
    printf("value（初期状態）：（10進数）%8d,（16進数）%08X\n", value, value);

    for (i = bytes - 2; i >= 0; i--)
    {
        value <<= 8;
        printf("value（シフト後）：（10進数）%8d,（16進数）%08X\n", value, value);
        value += ary[lsb_index + i];
        printf("value（加算後）　：（10進数）%8d,（16進数）%08X\n", value, value);
    }

    return (value);
}
```

ex_03_ref_1(result)

```
＜配列＞
ary[0]：（10進数）239,（16進数）EF
ary[1]：（10進数）205,（16進数）CÐ
ary[2]：（10進数） 35,（16進数）23
ary[3]：（10進数）  1,（16進数）01

＜途中経過＞
value（初期状態）：（10進数）       1,（16進数）00000001
value（シフト後）：（10進数）     256,（16進数）00000100
value（加算後）  ：（10進数）     291,（16進数）00000123
value（シフト後）：（10進数）   74496,（16進数）00012300
value（加算後）  ：（10進数）   74701,（16進数）000123CÐ
value（シフト後）：（10進数）19123456,（16進数）0123CÐ00
value（加算後）  ：（10進数）19123695,（16進数）0123CÐEF

＜結果＞
（10進数）19123695,（16進数）0123CÐEF
```

最後に，参考として，今回の問題について，数式を使って解釈した例を示しておく．よく理解しておこう．

（例）

16,875,648

$$
\begin{aligned}
&= 0\times 2^{31} + \cdots + 1\times 2^{24} + 0\times 2^{23} + \cdots + 1\times 2^{16} + 1\times 2^{15} + \cdots + 0\times 2^{8} + 1\times 2^{7} + \cdots + 0\times 2^{0} \\
&= \left(0\times 2^{31} + \cdots + 1\times 2^{24}\right) + \left(0\times 2^{23} + \cdots + 1\times 2^{16}\right) + \left(1\times 2^{15} + \cdots + 0\times 2^{8}\right) + \left(1\times 2^{7} + \cdots + 0\times 2^{0}\right) \\
&= \left(0\times 2^{7} + \cdots + 1\times 2^{0}\right)\times 2^{24} + \left(0\times 2^{7} + \cdots + 1\times 2^{0}\right)\times 2^{16} \\
&\qquad + \left(1\times 2^{7} + \cdots + 0\times 2^{0}\right)\times 2^{8} + \left(1\times 2^{7} + \cdots + 0\times 2^{0}\right)\times 2^{0} \\
&= 1\times 2^{24} + 1\times 2^{16} + 128\times 2^{8} + 128\times 2^{0}
\end{aligned}
$$