［演習 04］　ビットマップファイル(2)

［演習の概要］

標準的な画像ファイルであるビットマップファイル（BMP ファイル，bitmap ファイル）のフォーマットを学び，ビットマップファイルにアクセスするプログラムを作成する．

［補足］

演習においては，必要に応じて，以下のファイルを各自の作業ディレクトリにコピーして用いること．

/home0/staff/www/IMG/test04.bmp	（画像サイズ4×4のBMPファイル）
/home0/staff/www/IMG/lenna.bmp	（画像サイズ512×512のBMPファイル）

演習で使用する BMP ファイルは，Finder ウィンドウなどでダブルクリックすると，「プレビュー」というアプリケーションソフトが起動して画像として見ることができる．必要に応じて，画像を見て，プログラムの作成や検証を行うこと．

なお，プログラムの作成においては，問題に指定されていなくても，必要に応じて，関数を追加で作成したり，マクロ定義を用いたりしても構わない．

【問題 4-A-1】

問題 3-B-1 のプログラムに追加修正を施し，画像データ部に格納されている画像データを追加表示するプログラムを作成せよ．ここで，

- 以下に示す main 関数を使用すること．
- 以下の表に示す関数 get_data を作成してこれを使用すること．
  （問題 3-B-1 で作成した関数 get_data を修正して使用すること．）
- 以下の表に示す関数 processing を作成してこれを使用すること．
- データ部の画像データを格納するための配列
  ```
  unsigned char imgin[3][512][512]
  ```
  を，グローバル変数として宣言すること．
  （演習では，各画素が R 信号 8 ビット，G 信号 8 ビット，B 信号 8 ビットからなる画像のみを使用し，512×512 より大きなサイズの画像は使用しないものとする．よって，配列の要素数は，3×512×512 として構わない．）
- データ部の画像データは，関数 get_data 内において，図 4.1 に示されるように，配列 imgin に格納すること．
- ヘッダ部のデータの表示は，関数 get_data 内において行うこと．
- 「画像の幅」，「画像の高さ」，「4 バイトアラインメントに伴う挿入バイト数」を格納する変数をグローバル変数として宣言しておき，関数 get_data 内において，これらの変数に値を代入しておくこと．

とする．

main 関数

```
int main(void)
{
  get_data();
  processing();

  return 0;
}
```

関数 get_data

形式	void get_data(void)
解説	任意の BMP ファイルのヘッダ部のデータと画像データ部のデータを読み込み，そのヘッダ部のデータを 16 進数で表示する．ここで，ファイルサイズ，オフセット，画像の幅，画像の高さ，1 画素当たりのビット数については，その意味する内容についても，10 進数で表示する．（表示内容・表示方法の詳細については，実行結果例を参照のこと．） また，4 バイトアラインメントを満たすために，ファイルの最後に何バイトのデータが挿入されているのかを求めて表示する．挿入バイト数は，ファイルサイズ filesize，オフセット offset，画像の幅 width，画像の高さ height，1 画素当たりのビット数 bits を用いて， filesize-offset-width*height*(bits/8) と求めることができる．
返却値	なし．

関数 processing

形式	void processing(void)
解説	配列 imgin に格納されている画像データを，R 信号，G 信号，B 信号ごとに 16 進数で表示する．ここで，画像サイズが width×height ならば，画像データの表示も width×height とすること． （表示内容・表示方法の詳細については，実行結果例を参照のこと．）
返却値	なし．

三元配列

```
for (y = height - 1; y >= 0; y--) {
for( x = 0;  x < width; x++ ) {
for (int i=2; i >=0; i--) {
imgin[i][x][y] =(unsigned char)
fgetc(fp);
}
}
}
```

画像左上端の画素の
R 信号

R 信号

imgin[0][0][0]	imgin[0][1][0]	imgin[0][2][0]	・・・
imgin[0][0][1]	imgin[0][1][1]		
imgin[0][0][2]			
・・・			

画像左上端の画素の
G 信号

G 信号

imgin[1][0][0]	imgin[1][1][0]	imgin[1][2][0]	・・・
imgin[1][0][1]	imgin[1][1][1]		
imgin[1][0][2]			
・・・			

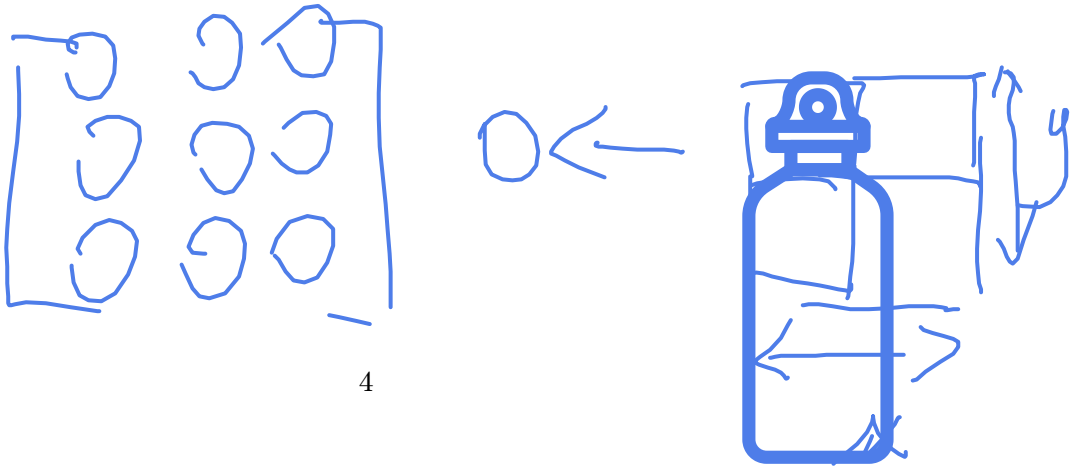
画像左上端の画素の
B 信号

B 信号

imgin[2][0][0]	imgin[2][1][0]	imgin[2][2][0]	・・・
imgin[2][0][1]	imgin[2][1][1]		
imgin[2][0][2]			
・・・			

上図は，画像の「1 番上のラインの左端の画素」の R 信号を，imgin[0][0][0]に格納するということを示している．図からわかるように，1 つ目の添字は信号の種類（R/G/B 信号）に，2 つ目の添字は水平方向の座標に，3 つ目の添え字は垂直方向の座標に対応している．

図 4.1　画像データの配列への格納方法



実行結果の例

```
入力ファイル名： test04.bmp return
test04.bmpをオープンしました．

                （中略．問題3-B-1と同様に出力すること．）

test04.bmpをクローズしました．

入力画像データを表示します．

＜R信号＞
00 01 02 03
04 05 06 07
08 09 0A 0B
0C 0Đ 0E 0F

＜G信号＞
10 11 12 13
14 15 16 17
18 19 1A 1B
1C 1Đ 1E 1F

＜B信号＞
20 21 22 23
24 25 26 27
28 29 2A 2B
2C 2Đ 2E 2F
```

【問題 4-B-1】

問題 4-A-1 のプログラムに追加修正を施し，ファイルのヘッダ部とデータ部のデータを表示した後，データ全体を他のファイルにコピーするプログラムを作成せよ．ここで，

- 以下に示す main 関数を使用すること．
- 関数 get_data 内において，ヘッダ部のデータを配列 header に，画像データ部のデータを配列 imgin に，一旦格納すること．（問題 4-A-1 で既にそのようになっている．）
- 以下の表に示す関数 processing を使用すること．
  （問題 4-A-1 で作成した関数 processing を修正すること．）
- 関数 processing 内において，配列 imgin に格納されているデータを，配列
  unsigned char imgout[3][512][512]
  にコピーすること．ここで，配列 imgout は，今後のプログラム作成の都合上，グローバル変数として宣言しておくこと．
- 以下の表に示す関数 put_data を作成してこれを使用すること．

とする．

main 関数

```
int main(void)
{
  get_data();
  processing();
  put_data();

  return 0;
}
```

関数 processing

形式	void processing(void)
解説	配列 imgin に格納されている画像データを，R 信号，G 信号，B 信号ごとに 16 進数で表示する．ここで，画像サイズが width×height ならば，画像データの表示も width×height とすること． （表示内容・表示方法の詳細については，実行結果例を参照のこと．なお，画像サイズが大きい場合には，ウィンドウ内で自動的に改行されてしまうので，width×height の大きさで適切に表示することができない．そのため，画像サイズが大きい場合には，データが表示されないようにする，あるいは，一部のデータのみが表示されるようにするなど，各自対応すること．） その後，配列 imgin に格納されている画像データを，配列 unsigned char imgout[3][512][512] にコピーする．
返却値	なし．

関数 put_data

形式	void put_data(void)
解説	配列 header に格納されているヘッダ部のデータと，配列 imgout に格納されているデータ部のデータを用いて，新しい BMP ファイルを作成する．新しいファイルの名前は，キーボードから入力させる． 新しい BMP ファイルは，4 バイトアラインメントを満たすものとする．すなわち，必要に応じて，ファイルの最後に何バイトかのデータを挿入すること．具体的には，画像データをすべて書き出した後に，4 バイトアラインメントを取るために必要なバイト数だけ fputc('¥0', fp); を行う．（データを何バイト挿入する必要があるかは，挿入バイト数として，get_data 内で既に求めているはず．）
返却値	なし．

実行結果の例(1)

```
入力ファイル名： test04.bmp return
test04.bmpをオープンしました．

                （中略．問題3-B-1と同様に出力すること．）

test04.bmpをクローズしました．

入力画像データを表示します．

                （中略．問題4-A-1と同様に出力すること．）

出力画像データを作成しました．

出力ファイル名： test04cp.bmp return
test04cp.bmpをオープンしました．
test04cp.bmpをクローズしました．
```

実行結果の例(2)

```
入力ファイル名： lenna.bmp return
lenna.bmpをオープンしました．

                （中略．問題3-B-1と同様に出力すること．）

lenna.bmpをクローズしました．

入力画像データを表示します．

            （中略．画像データが大きい場合への対応が行われていること．）

出力画像データを作成しました．

出力ファイル名を入力して下さい： lennacp.bmp return
lennacp.bmpをオープンしました．
lennacp.bmpをクローズしました．
```

ここで，作成された lennacp.bmp をダブルクリックすると，プレビューが起動して画像を見ることができる．必ず確認すること．（画像が表示されない場合は，プログラムに誤りがある．）