Chapter 6

画像を操作する

画像のフォーマットは、tiff、gif、jpeg、pngなどいろいろあるけれど、比較的構造が単純で扱いやすいビットマップ(Bitmap)形式の画像ファイルを扱うプログラムを作ってみよう。

6.1　ビットマップ

ビットマップは、赤色(r)、緑色(g)、青色(b)を1画素として、これを並べて画像を構成する。表6.1は、ビットマップ画像における画素の並び順を表している。画素は、画像ファイルの左下を原点とし、右に画素が並び横幅分に達すると、一行上に並ぶ。このような並び方で画素は画像ファイルの一番上の行まで並ぶ。画像ファイルに記録されているデータとして、先頭にヘッダ情報[1]の値が、その後に各画素に対応した色の情報が青色(b)、緑色(g)、赤色(r)の順番に値が並んでいる。

imageutil.hとimageutil.c

imageutil.hにこれらの情報を保持するための構造体を定義する。PIXEL構造体は1画素の情報を、IMAGE構造体は画像のタテ、ヨコ、深さ（=画像の色数)、PIXEL構造体の配列[2]をそれぞれメンバーとしている。witdhメンバーの値が100である場合、IMAGE構造体のPIXEL型の配列の0番目の

[1]ビットマップ画像のタテ・ヨコのサイズや色数などの情報が格納される。細々した話でもあるのでこの演習ではこれ以上深入りしない。気になる人は是非いろいろ調べてみてほしい。

[2]ただし、ポインタとして実現している。

Table 6.1: ビットマップの画素の並び方

$(h-1)w$	$(h-1)w+1$	$(h-1)w+2$	$\cdots$	$hw-1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2w$	$2w+1$	$2w+2$	$\cdots$	$3w-1$
w	$w+1$	$w+2$	$\cdots$	$2w-1$
0	1	2	$\cdots$	$w-1$

要素は画像の左端の画素を指し、502番めの要素は下から6行目左から3列目の画素を指すことになる。imgutil.hは、以下の通り。

```
#ifndef IMGUTIL_H_
#define IMGUTIL_H_
#include <stdio.h>

struct pixel{
  int r, g, b;
};

typedef struct pixel PIXEL;

struct image{
  int width, height, depth;
  PIXEL *pixels;
};

typedef struct image IMAGE;

long int getLabel(int x, int y, int width);
#endif /* IMGUTIL_H_ */
```

このとき、このヘッダファイルでプロトタイプ宣言されているgetLabel関数をimgutil.cファイルに定義せよ。

- long int getLabel(int x, int y, int width);

  画像左下の画素を原点$(0,0)$とし、右側をx軸（右向きが正）、上側をy軸（上向きが正）とする。また、画像の横幅がwidthピクセル

あるとする。このとき、表6.1のように画素に順番をつけると、座標(x, y)にある画素は何番目の画素に相当するかを計算せよ。この関数はこの結果（該当する画素が何番目か）を返すこととする。

word.hとword.c

通常、コンピューター上で色を表すときにはRGB色空間というものを使う。これは6.1でも述べたように赤色(r)、緑色(g)、青色(b)で画素を表現するものだ。しかし、ビットマップ[3]では、各画素について青色(b)、緑色(g)、赤色(r)の順番にファイルに値が並べられている。ちょうどRGB色空間の値の並びと逆の並びになっているわけだ。

ビットマップのヘッダの値についても、8bit(=1byte)ずつ下位ビットから順番にメモリーに書き込まれていく必要がある。しかし、各データを都度、並べ直す処理を用意するのは無駄である。そこで、これも関数にしてしまおう。まず、word.hというヘッダファイルを用意しよう。8bitはちょうどchar型のデータと同じサイズなので、書き込みにはfputc関数を使うとよい。

```
#ifndef WORD_H_
#define WORD_H_
#include <stdio.h>

typedef unsigned short WORD; /* 符号なしshort intをWORDと呼ぶ*/
typedef unsigned long DWORD; /* 符号なしlong intをDWORDと呼ぶ*/
typedef unsigned char BYTE; /* 符号なしcharをBYTEと呼ぶ*/

void fwriteWORD(WORD w, FILE *fp);
void fwriteDWORD(DWORD dw, FILE *fp);

#endif /* WORD_H_ */
```

ここにあるfwriteWORD関数、fwriteDWORD関数を以下のように定義する。これを実装しよう。（ファイルはword.cとする。）

- void fwriteWORD(WORD w, FILE *fp)

  引数wを256(=8bit)で割った余り求め、ファイルに書き出す。wを256で割った商をwに代入する。この操作を2回繰り返す。なお、ファイルの書き出しにはFILEポインタを利用せよ。

[3]少なくとも非圧縮24bit形式のビットマップは、という意味。

- void fwriteDWORD(DWORD dw, FILE *fp)

  引数dwを256(=8bit)で割った余り求め、ファイルに書き出す。dwを256で割った商をdwに代入する。この操作を4回繰り返す。なお、ファイルの書き出しにはFILEポインタを利用せよ。

export.hとexport.c

いよいよ、ビットマップデータを画像ファイルとして出力するための関数を作ろう。まず、export.hは以下のように定義する。

```
#ifndef EXPORT_H_
#define EXPORT_H_

#include "word.h"
#include "imgutil.h"
#include <stdio.h>

int saveImage(FILE *fp, IMAGE *img);

#endif /* EXPORT_H_ */
```

ここで宣言されているsaveImage関数を次のように実装したい（ファイル名はexport.c)。以下には既にヘッダをファイルに書き込む処理は記述してある[4]。

なお、下記のsaveImage関数の中の処理で“Writing data...”と表示する処理の下にコメント行があるが、その部分にIMAGE構造体のポインタ変数imgのpixelsメンバーはPIXEL構造体の配列（ポインタ）であることを思い出し[5]、左下端の画素に相当するpixels配列の要素から順番に、青(b)、緑(g)、赤(r)の順に値をファイルに書出せ。ただし、青、緑、赤は0から255までの値をとるため、char型の変数で取り扱える範囲の値であることに注意し、ファイルへの出力はfputc関数を用いること。

```
#include "word.h"
#include "imgutil.h"
#include "export.h"
```

[4] word.cで実装したfwriteWORD関数とfwriteDWORD関数が使われていることに注目されたし。

[5] はて？と思ったひとは、imgutil.hを見直すべし。

```
int saveImage(FILE *fp, IMAGE *img){

  WORD bfType=0x4d42; /* BM */
  DWORD bfSize=40;
  WORD bfReserved1=0;
  WORD bfReserved2=0;
  DWORD bfOffset=54;
  DWORD biSize=40;
  DWORD biWidth=img->width;
  DWORD biHeight=img->height;
  WORD biPlanes=1;
  WORD biBitCount=img->depth;
  DWORD biCompression=0;
  DWORD biSizeImage=0;
  DWORD biXPelsPerMeter=300;
  DWORD biYPelsPerMeter=300;
  DWORD biClrUsed=0;
  DWORD biClrImportant=0;
  int x,y,i=0;
  PIXEL p;

    printf("Start.\n");
    /* This program supports only 24bit depth for simplicity.*/
    if(img->depth!=24){
      printf("Sorry, this supports only 24bit depth.\n");
      return 0;
    }
    printf("Writing header...\n");
    fwriteWORD(bfType, fp);
    fwriteDWORD(bfSize, fp);
    fwriteWORD(bfReserved1, fp);
    fwriteWORD(bfReserved2, fp);
    fwriteDWORD(bfOffset, fp);
    fwriteDWORD(biSize, fp);
    fwriteDWORD(biWidth, fp);
    fwriteDWORD(biHeight, fp);
    fwriteWORD(biPlanes, fp);
```

```
    fwriteWORD(biBitCount, fp);
    fwriteDWORD(biCompression, fp);
    fwriteDWORD(biSizeImage, fp);
    fwriteDWORD(biXPelsPerMeter, fp);
    fwriteDWORD(biYPelsPerMeter, fp);
    fwriteDWORD(biClrUsed, fp);
    fwriteDWORD(biClrImportant, fp);
    printf("Writing data...\n");
    /*
      imgのpixelsメンバが各ピクセルの色データを
      持っている配列であることを思い出し、各ピクセルのデータを順
に
      青(b)、緑(g)、赤(r)の順番にファイルに書出せ。

      青(b)、緑(g)、赤(r)ともに0から255までの値をとるので
      char型の変数で取り扱えるデータである。そのため、
      ファイルへの書き出しにはfputc関数を使うと良い。
    */
    printf("done!\n");

    return 1;
}
```

実行するためのプログラム

以上の関数が出来上がれば、これらを使って実際にビットマップファイルを出力してみよう。今回作ったexport.c、imgutil.c、word.cと下に示すtest.cをコンパイル・リンクして実行ファイルcircleGeneratorを作成し、実行してみよ。ただし、実行前には必ず下のソースを見て、どのようなビットマップファイルが出来上がるか予測してみること。

```
/*
 * test.c
 *
 *  Created on: 2013/03/31
 *      Author: masaomi
 */

#include "imgutil.h"
#include "export.h"
#include <stdio.h>
#include <stdlib.h>

int graph(int x, int y);

int main(void){

  FILE *fp;
  IMAGE *img=(IMAGE *)malloc(sizeof(IMAGE));
  int ret,x=0,y=0;
  int height=120, width=120;
  PIXEL *pixels=(PIXEL *)malloc(height*width*sizeof(PIXEL));
  img->height=height;
  img->width=width;
  img->depth=24;
  img->pixels=pixels;
  long int label;

  label=0;
  for(y=img->height-1; y>=0; y--){
    for(x=0; x<img->width; x++){
    label=getLabel(x,y,img->width);
```

```
      if(graph(x,y)){
        img->pixels[label].r=0;
        img->pixels[label].g=0;
        img->pixels[label].b=255;
      }else{
        img->pixels[label].r=255;
        img->pixels[label].g=255;
        img->pixels[label].b=255;
      }
    }
  }

  printf("Save data as a file, check.bmp.\n");
  fp=fopen("check.bmp", "w");
  printf("Save!\n");
  ret=saveImage(fp, img);
  if(!ret){
    printf("ERROR -- could not write the image.");
    return 0;
  }
  printf("done.");
  fclose(fp);
  return 0;
}

int graph(int x, int y){
  if((x-60)*(x-60)+(y-60)*(y-60)>800 &&(x-60)*(x-60)+(y-60)*(y-60)<1000){
    return 1;
  }else{
    return 0;
  }
}
```