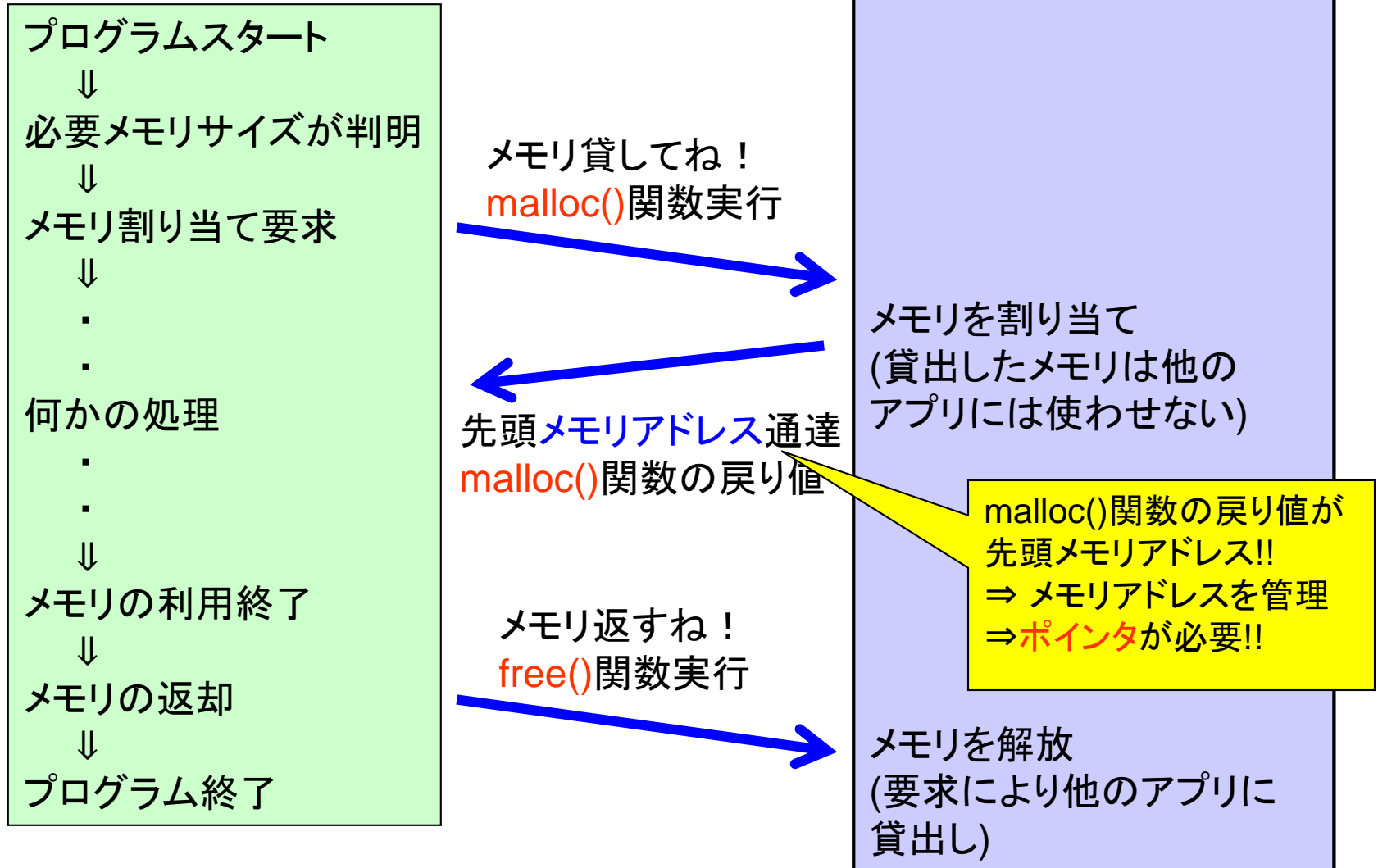


メモリの動的割り当ての概念

画像を処理するプログラム
(アプリケーションプログラム)

OS (Windows, iOS, Android, Linux 他)



変数のアドレスとポインタ(1)

```
int a = 100;
printf("値は%dで, アドレスは%dである\n", a, &a)
```

値は100で, アドレスは1245052である

ルール1

&a は変数 **a** の先頭
アドレスを意味する

int型 **ポインタ変数 p** を宣言

```
int a = 100;
int *p;
p = &a;
```

ルール2

ポインタ変数の宣言では
変数名の前に ***** を付ける

ポインタ変数 **p** に変数 **a** のアドレスを代入

```
printf("値は%dで, アドレスは%dである\n", a, p)
```

値は100で, アドレスは1245052である

変数のアドレスとポインタ(3)

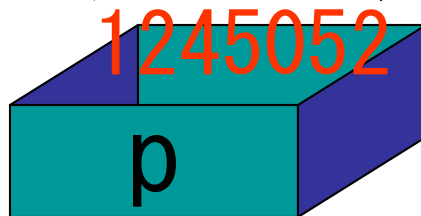
```
int a = 100;
int *p;
p = &a;
printf("値は%dで、アドレスは%dである\n", *p, p)
```

値は100で、アドレスは1245052である

ルール3

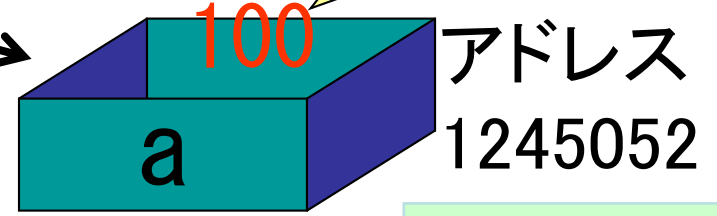
ポインタ変数名の前に ***** を付けると、指し示している変数に入っている値を意味する

p の値はアドレス



ポインタ変数

***p** の値は100



変数

***p** \leftrightarrow **a**
同じ値

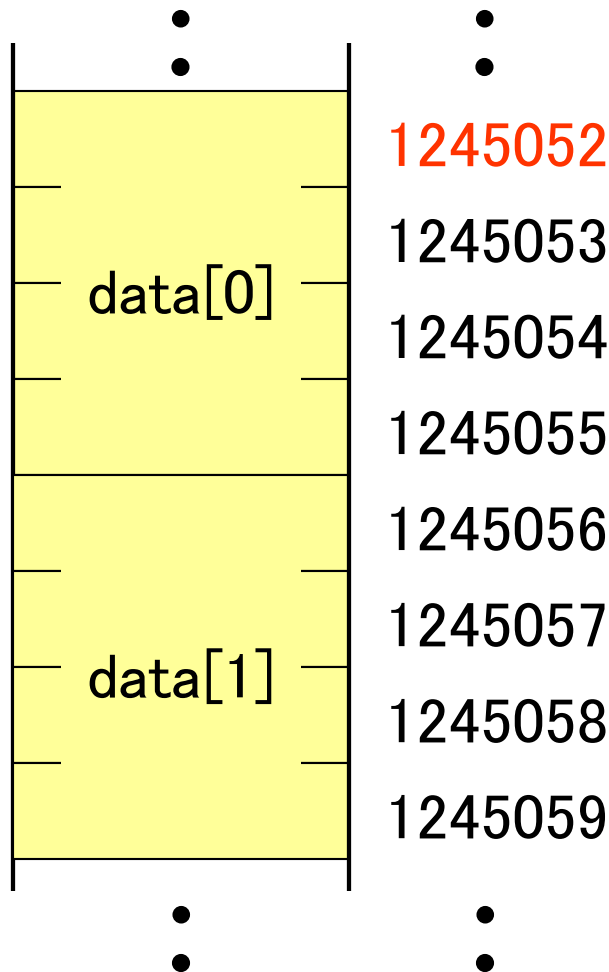
1次元配列のアドレスとポインタ

```
int data[10];
int *p;
p = &data[0];
```



```
int data[10];
int *p;
p = data;
```

この場合、&は
いらない



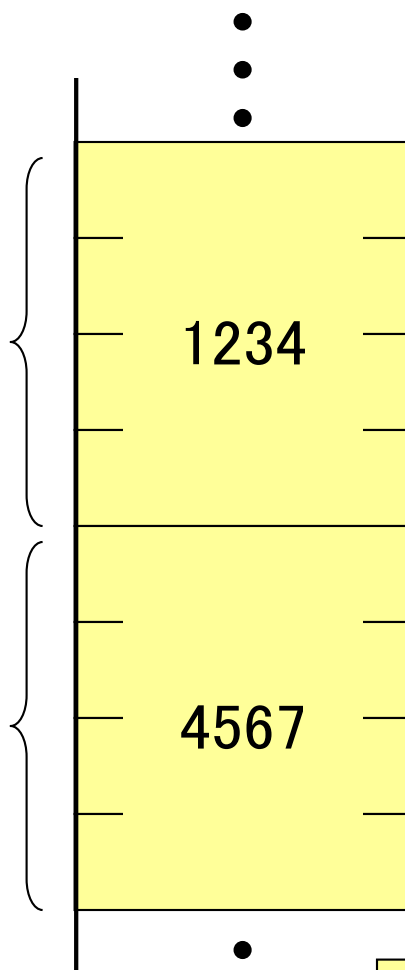
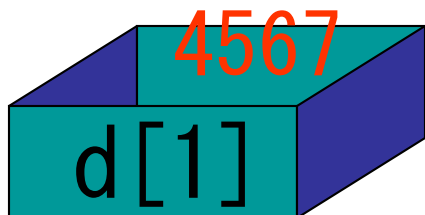
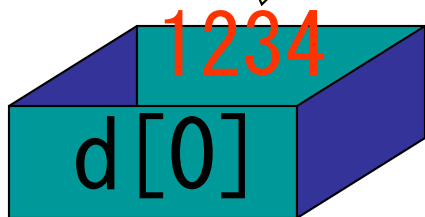
配列の先頭
アドレス
`&data[0]`
または
`data`

ルール4
配列の名前だけ
を書くと、そ
れは配列の先
頭アドレスを
意味する

ポインタと1次元配列の関係

```
int d[10];
int *p;
p = d;
```

int型配列



0002364

0002365

0002366

0002367

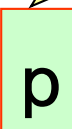
0002368

0002369

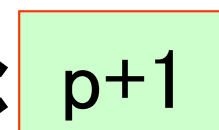
0002370

0002371

int型ポインタ



$*p \Leftrightarrow d[0] = 1234$



$*(p+1) \Leftrightarrow d[1] = 4567$

変数値

`d[0]` \longleftrightarrow `*p`

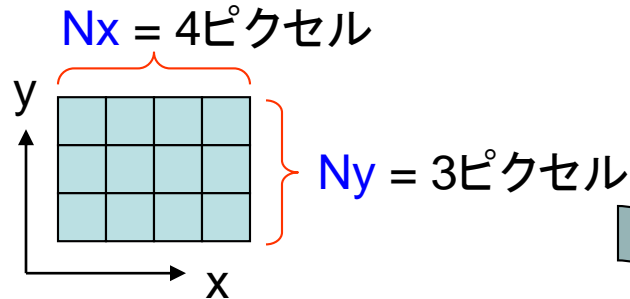
`d[1]` \longleftrightarrow `*(p+1)`

`d[2]` \longleftrightarrow `*(p+2)`

どちらを使っても全く同じ!

2次元配列で表した画像とポインタの関係

次の様に座標を決めた画像を・・・



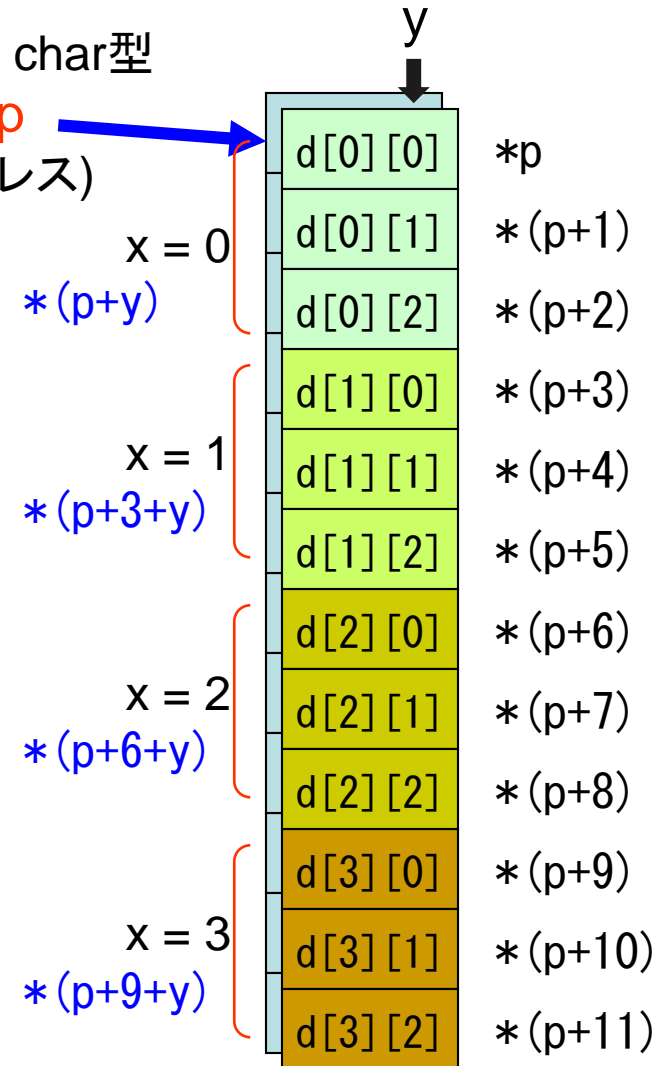
下の様な配列で考えると・・・

```
unsigned char d[4][3];
```

$y = 2$	$d[0][2]$	$d[1][2]$	$d[2][2]$	$d[3][2]$
$y = 1$	$d[0][1]$	$d[1][1]$	$d[2][1]$	$d[3][1]$
$y = 0$	$d[0][0]$	$d[1][0]$	$d[2][0]$	$d[3][0]$
	$x = 0$	$x = 1$	$x = 2$	$x = 3$

$d[x][y] \rightarrow *(p + x*N_y + y)$

unsigned char型
ポインタ p
(先頭アドレス)



注) 動的割り当てしたメモリを2次元配列の形でソース中で参照できない

動的割り当てとポインタを用いた例

Example3-1

```
#include <stdio.h>
#include <stdlib.h>
#include "cglec.h"
int main(void)
{
    int Nx, Ny;
    printf("画像の横方向ピクセル数は? "); scanf("%d", &Nx);
    printf("画像の縦方向ピクセル数は? "); scanf("%d", &Ny);

    unsigned char* data;
    data = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
    if (data == NULL)
    {
        printf("メモリエラー!!");
        exit(0);
    }

    Image img = { (unsigned char*) data, Nx, Ny };
    CglSetAll(img, 0);
    int x, y;
    for (y = Ny/4; y < Ny/2; y++)
        for (x = Nx/4; x < Nx/2; x++)
        {
            *(data + x*Ny + y) = 255;
        }
    CglSaveGrayBMP(img, "Rei3-1.bmp");
    free(data);
}
```

unsigned char型のポインタ

unsigned char型ポインタへの変換

malloc(sizeof(unsigned char) * Nx * Ny);

//メモリ割当てに失敗か?

printf("メモリエラー!!");

exit(0);

//プログラムを終了する

stdlib.h中で宣言されたマクロ

#define NULL 0

2次元配列で書き直すことはできない

data[x][y] = 255; ✕

//メモリ解放

void* malloc(size_t s)
ヘッダファイル stdlib.h
引数 s: 割り当てるメモリのバイト数
 注) size_t型はほぼint型と同じ
戻り値 メモリの先頭へのポインタ
 注) void型ポインタは不明な変数型へのポインタなので、何かの型へのポインタにキャストしないとポインタとして使用できない
戻り値がNULL(= 0)の場合
 割り当てに失敗

sizeof(「データ型」)
 「データ型」の変数一つに必要なメモリのバイト数を返す **演算子**
 注)
 sizeof(unsigned char) ⇒ 1
 従って簡易的に書くなら
 . . . malloc(Nx*Ny) . . .
 としても良い。

void free(void* mem)
ヘッダファイル stdlib.h
引数 mem: 解放するメモリの先頭アドレス
戻り値 なし

動的割り当ての成功・失敗の判定は必ず必要

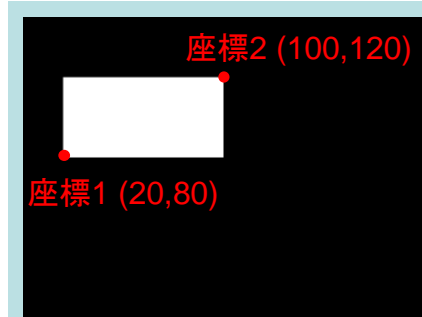
理由: メモリ不足などの理由で割り当てに失敗する可能性がある。

基本課題3

復習

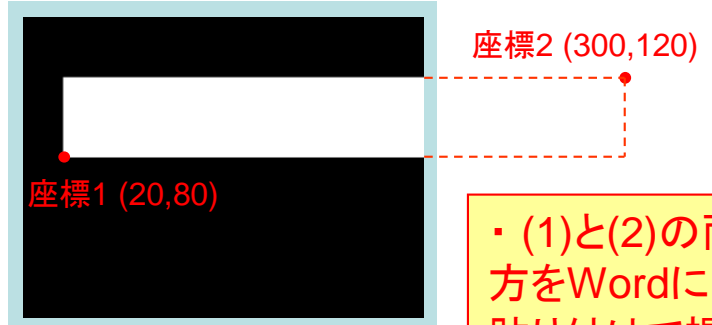
画像サイズを入力し、次に四角形の左下の座標(x_1, y_1)と右上の座標(x_2, y_2)を入力するとその四角形を白色で塗りつぶすプログラムを作成せよ。ただし(x_2, y_2)の位置が画像の外側である場合は画像の範囲内で四角形を描くこと。

画像の横方向ピクセル数は？ 200
画像の縦方向ピクセル数は？ 150
X1 = ? 20
Y1 = ? 80
X2 = ? 100
Y2 = ? 120



X1とY1は画像の内側の正しい値だと仮定して良い

画像の横方向ピクセル数は？ 200
画像の縦方向ピクセル数は？ 150
X1 = ? 20
Y1 = ? 80
X2 = ? 300
Y2 = ? 120



X2とY2が不正な値である場合も正しく処理すること！

画像の横方向ピクセル数は？ 200
画像の縦方向ピクセル数は？ 150
X1 = ? 20
Y1 = ? 80
X2 = ? 300
Y2 = ? 200

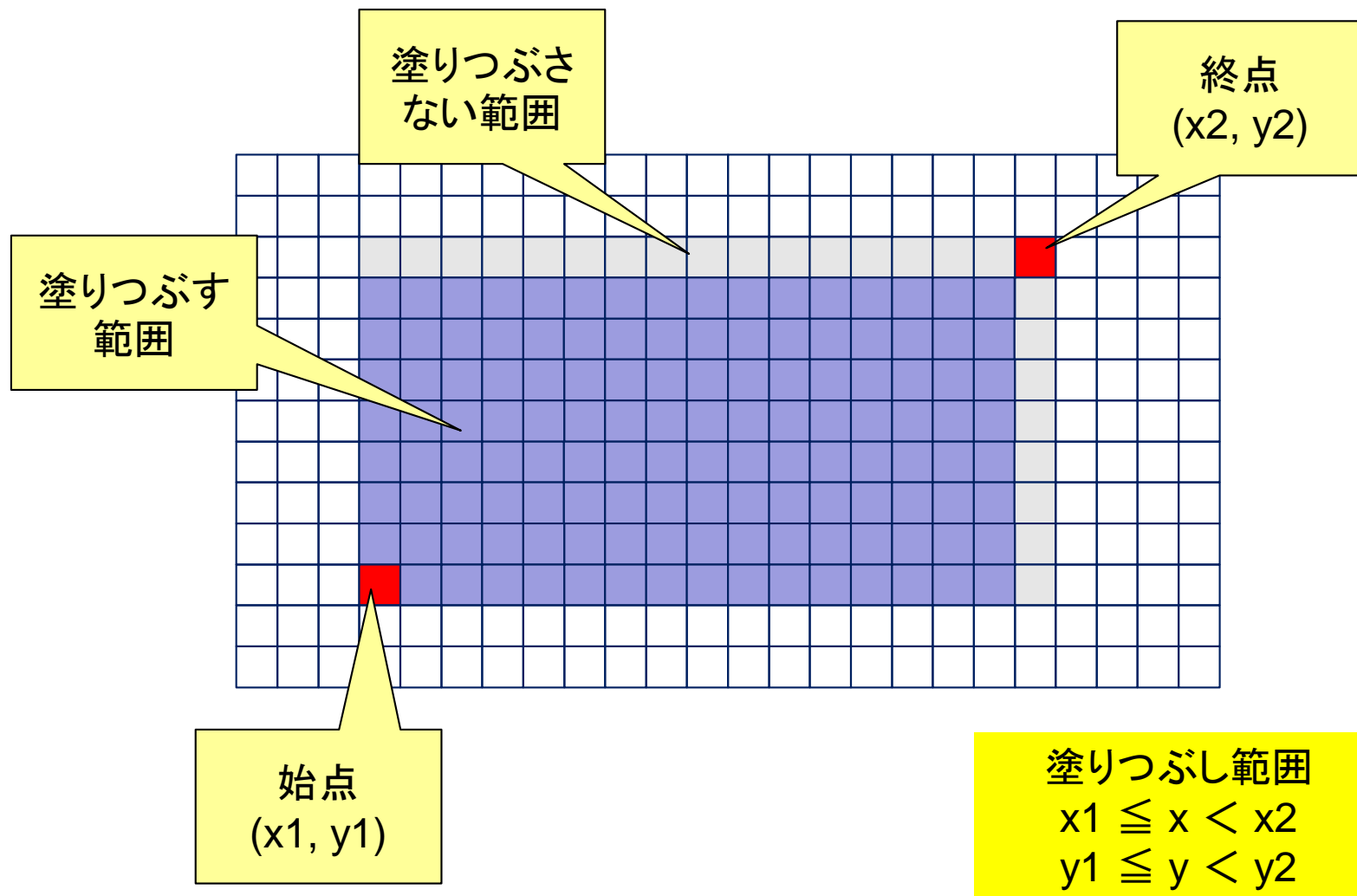


- ・ (1)と(2)の両方をWordに貼り付けて提出！
- ・ 実行例を二つ以上提出！

(1) 実行結果 (スクリーンショット)

(2) 実行結果(BMPファイル)

図形範囲指定の補足



未完成レポートについて

- プログラムが**未完成**あるいは**動作が不完全**であるにもかかわらず、**完成したかを装ったような悪意のあるレポート**を出した場合は、**マイナス点**とします。
- 課題プログラムが**未完成**あるいは**不完全動作**に終わった場合は、レポートの冒頭で目立つように「**未完成です**」という表記を入れること。
 - 未完成であっても、**わずかな努力点を加点**する場合があります。

悪意のある未完成レポートの例

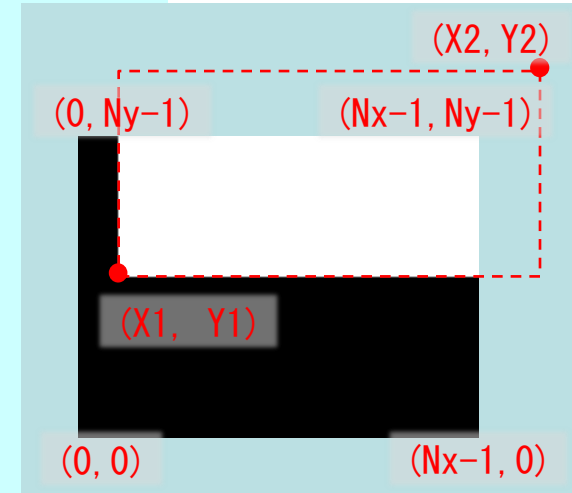
- ソースと実行結果が一致しない ⇒ マイナス点を加点
- 特定の実行例ではエラーになって実行できないにも関わらず、それを隠している ⇒ マイナス点を加点

A君解答

基本課題3 解答例

```
#include <stdio.h>
#include <stdlib.h>
#include "cglec.h"
int main(void)
{
    int Nx, Ny;
    int X1, Y1, X2, Y2;
    printf("画像の横方向ピクセル数は?"); scanf("%d", &Nx);
    printf("画像の縦方向ピクセル数は?"); scanf("%d", &Ny);
    printf("X1= "); scanf("%d", &X1);
    printf("Y1= "); scanf("%d", &Y1);
    printf("X2= "); scanf("%d", &X2);
    printf("Y2= "); scanf("%d", &Y2);
```

```
    unsigned char* data;
    data = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
    if (data == NULL) //メモリ割当に失敗か?
    {
        printf("メモリエラー!!");
        exit(0); //プログラムを終了する
    }
    Image img = { (unsigned char*) data, Nx, Ny };
    CglSetAll(img, 0);
```



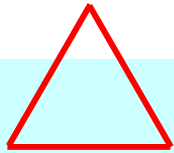
```
    if(X2 > Nx) X2 = Nx;
    if(Y2 > Ny) Y2 = Ny;

    int x, y;
    for (y = Y1; y < Y2; y++)
        for (x = X1; x < X2; x++)
        {
            *(data + x*Ny + y) = 255;
        }
    CglSaveGrayBMP(img, "Rei3-1.bmp");
    free(data); //メモリ解放
}
```

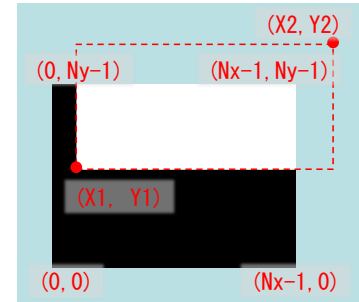
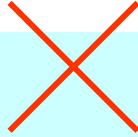
基本課題3 解答例

非効率な例

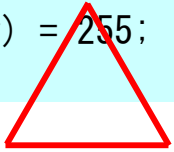
```
int x, y;
for(y=Y1; y<Y2; y++) {
    for(x=X1; x<X2; x++) {
        if(X2>Nx) { X2=Nx;};
        if(Y2>Ny) { Y2=Ny;};
        *(data + x*Ny + y) = 255;
    }
}
```



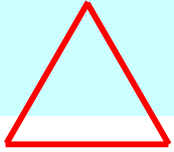
```
int x=0, y=0;
for(y=y1; y<=y2; y++) {
    for(x=x1; x<=x2; x++) {
        if(x<=Nx && y<=Ny)
            *(str+(Ny*x)+y)=255;
    }
}
```



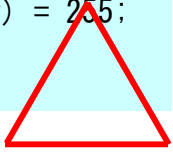
```
for (y = y1; y < y2 && y < Ny ; y++)
    for (x = x1; x < x2 && x < Nx ; x++)
    {
        *(data + x*Ny + y) = 255;
    }
```




```
for (y = Ny1; y < Ny; y++)
{
    for (x = Nx1; x < Nx; x++)
    {
        *(data + x*Ny + y) = 255;
        if (x == Nx2)
            break;
    }
    if (y == Ny2)
        break;
}
```



```
for (y = y1; y < y2; y++)
{
    if (y == Ny)
        break;
    for (x = x1; x < x2; x++)
    {
        if (x == Nx)
            break;
        *(data + x*Ny + y) = 255;
    }
}
```



```
for (y = Y1; y < Y2; y++)
{
    if (y == Ny)
        break;
    for (x = X1; x < X2; x++)
    {
        *(data + x*Ny + y) = 255;
        if (x == Nx)
            break;
    }
}
```



効率的なプログラムの条件

- 1) ループを実行する前にしっかりと事前処理する
パラメータチェック！
- 2) ループ内での処理をできるだけ簡単化する
条件判断(if文)がループに入るとスピードダウン！
- 3) ループの回数をできるだけ減らす
アルゴリズムの検討！

発展課題3

復習

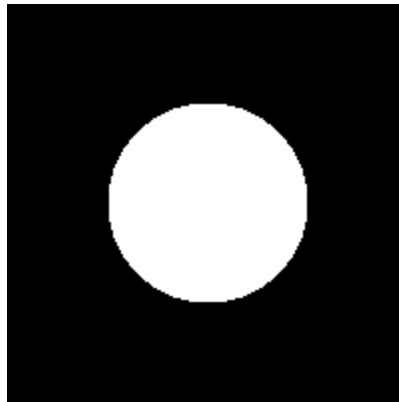
次回より円の塗りつぶしプログラムを学習するので、予習のためにぜひやってみよう。

画像サイズを入力し、その短辺の2分の1を直径とする円を白で塗りつぶすプログラムを作成せよ

画像の横方向ピクセル数は？ 200
画像の縦方向ピクセル数は？ 100



画像の横方向ピクセル数は？ 200
画像の縦方向ピクセル数は？ 200



画像の横方向ピクセル数は？ 50
画像の縦方向ピクセル数は？ 400



発展課題3 ヒント

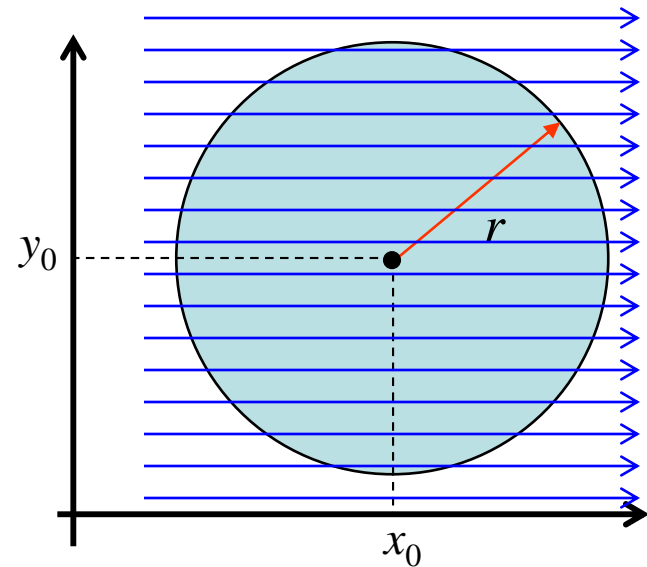
(x_0, y_0) を中心とする半径 r の円の方程式は

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

である. 従って, この円の内側の領域は

$$(x - x_0)^2 + (y - y_0)^2 \leq r^2$$

で表される.

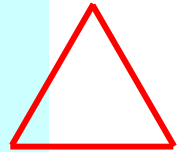


ラスタースキャン

発展課題3 解答例

C君解答

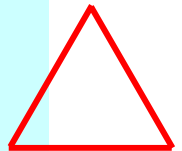
```
if (Nx > Ny) { r = Ny / 4; }
else { r = Nx / 4; }
CglSetAll(img, 0);
int x, y;
for (y = 0; y < Ny; y++)
    for (x = 0; x < Nx; x++)
    {
        if ((x-Nx/2)*(x - Nx / 2) + (y-Ny/2)*(y - Ny / 2) <= r*r)
            *(data + x*Ny + y) = 255;
    }
CglSaveGrayBMP(img, "Rei3-1.bmp");
```



D君解答

```
CglSetAll(img, 0);
int x, y;

for (y = 0; y < Ny; y++)
    for (x = 0; x < Nx; x++)
    {
        if (Nx < Ny)
        {
            if ((x-Nx/2)*(x-Nx/2) + (y-Ny/2)*(y-Ny/2) <= Nx/4*Nx/4)
                *(data + x*Ny + y) = 255;
        }
        else
        {
            if ((x-Nx/2)*(x-Nx/2) + (y-Ny/2)*(y-Ny/2) <= Ny/4*Ny/4)
                *(data + x*Ny + y) = 255;
        }
    }
CglSaveGrayBMP(img, "Rei3-1.bmp");
```



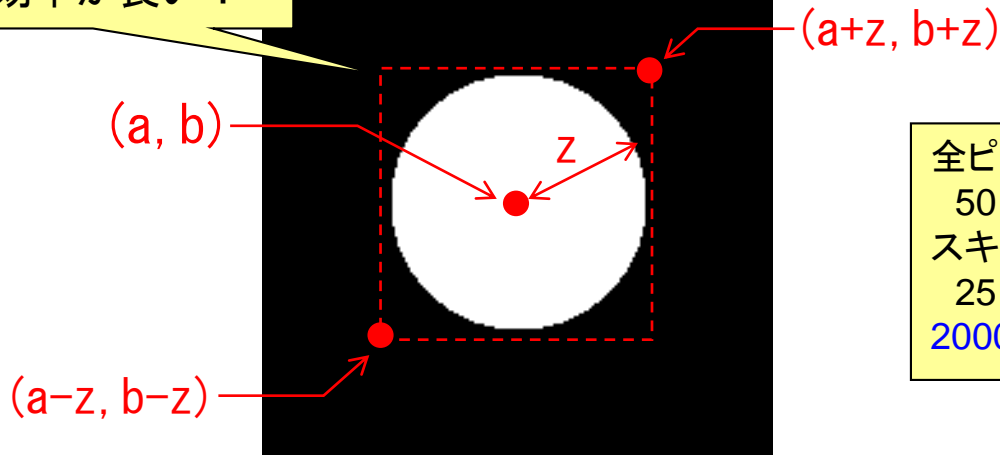
発展課題3 解答例

E君解答

```
CglSetAll(img, 0);
int a=Nx/2, b=Ny/2, x, y, z;
if(Nx<Ny)
    z=a/2;
else
    z=b/2;
for (x=a-z; x<a+z; x++)
{
    for (y=b-z; y<b+z; y++)
    {
        if ((x-a)*(x-a)+(y-b)*(y-b)<=z*z)
            *(data + x*Ny+ y) = 255;
    }
}
CglSaveGrayBMP(img, "Rei3-1.bmp");
```

良くできました!

この範囲だけをスキャンするので効率が良い!



全ピクセル数
 $50 \times 400 = 20,000$
 スキャン範囲
 $25 \times 25 = 625$
 $20000/625 = 32$ 倍!

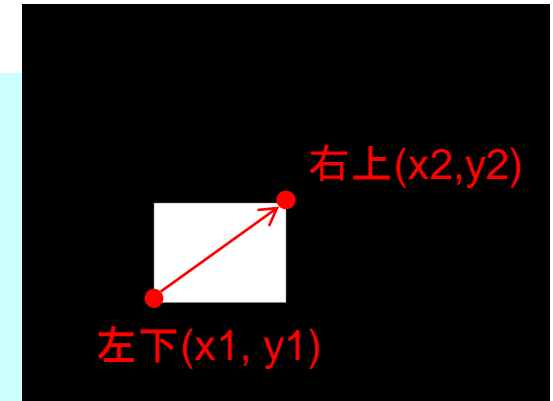
関数を作ろう！

Example4-1

```
#include <stdio.h>
#include <stdlib.h>
#include "cglec.h"
int main(void)
{
    int Nx, Ny;
    printf("画像の横方向ピクセル数は？ "); scanf("%d", &Nx);
    printf("画像の縦方向ピクセル数は？ "); scanf("%d", &Ny);

    unsigned char* data;
    data = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
    if (data == NULL) //メモリ割当に失敗か？
    {
        printf("メモリエラー!!");
        exit(0); //プログラムを終了する
    }

    Image img = { (unsigned char*) data, Nx, Ny };
    GglSetAll(img, 0);
    int x, y;
    for (y = Ny/4; y < Ny/2; y++)
        for (x = Nx/4; x < Nx/2; x++)
        {
            *(data + x*Ny + y) = 255;
        }
    GglSaveGrayBMP(img, "Rei3-1.bmp");
    free(data); //メモリ解放
}
```



四角形の描画はよく利用するので、関数にしておこう！

左下の点座標

右上の点座標

PaintRect(img, x1, y1, x2, y2, 255);

Image型構造体変数
↓
描画する「キャンバス」
を指定する

四角形描画関数

Example4-2

```
void PaintRect(Image img, int x1, int y1, int x2, int y2, int level)
```

```
{
    int temp;
    if (x1 > x2)    // x1 < x2 になるように入れ替え
    {
        temp = x1;  x1 = x2;    x2 = temp;
    }
    if (y1 > y2)    // y1 < y2 になるように入れ替え
    {
        temp = y1;  y1 = y2;    y2 = temp;
    }
}
```

// パラメータが範囲外の場合は修正


```
if (x1 < 0)        x1 = 0;
if (y1 < 0)        y1 = 0;
if (x2 > img.Nx)   x2 = img.Nx;
if (y2 > img.Ny)   y2 = img.Ny;
if (level < 0)     level = 0;
if (level > 255)   level = 255;
```

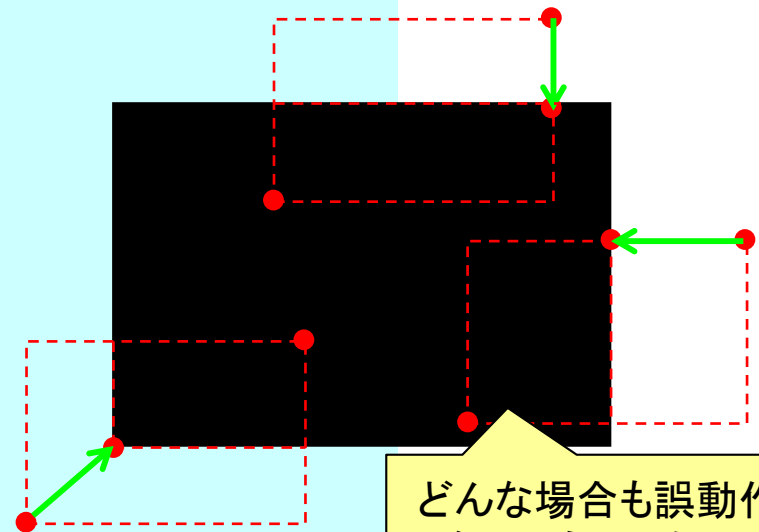
// 四角形描画

```
int x, y;
for (y = y1; y < y2; y++)
    for (x = x1; x < x2; x++)
    {
        *(img.Data + x*img.Ny + y) = level;
    }
}
```

構造体変数imgの画像に対する描画なので、画像の幅や高さはimgのメンバー変数を用いなければならない

機能

画像において、
二点(x1, y1), (x2, y2)を対
角点とする四角形をグレー
レベルlevelで塗りつぶす



どんな場合も誤動作しないようにしたい！

```
struct Image
{
    unsigned char* Data; // データ領域へのポインタ
    int Nx;              // 横方向ピクセル数
    int Ny;              // 縦方向ピクセル数
};
```

基本課題4

中心座標(x_0, y_0)と半径 r , グレーレベル g を引数として, 円を塗りつぶす関数を作成せよ.
この時, 引数が適切な範囲で無い場合も正常に動作すること. 作成した関数と下記の
main()を組み合わせて実行例と同じ画像を得よ. **main()関数を修正してはいけない!**

Report4-1

```
#include <stdio.h>
#include <stdlib.h>
#include "cglec.h"

void PaintCircle(Image img, int x0, int y0, int r, int g)
{
    // この関数を作成する.  x0, y0 : 中心座標,  r : 半径,  g : グレーレベル
}

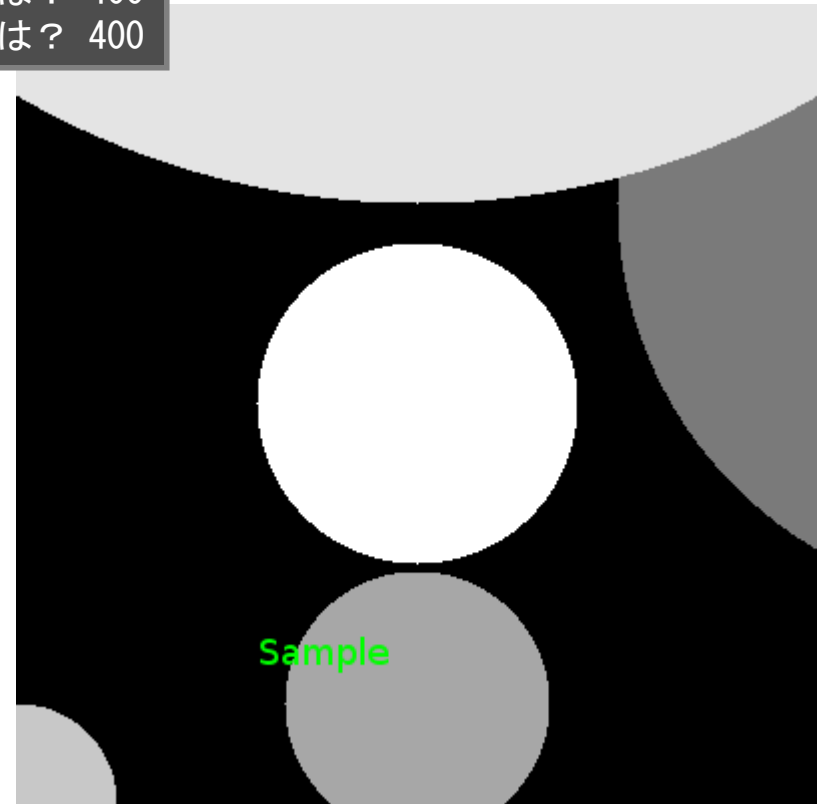
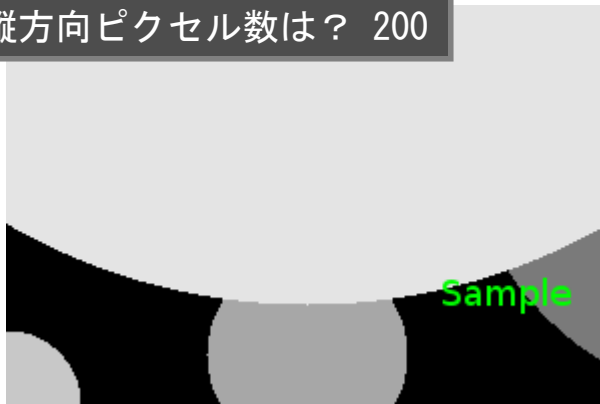
int main(void)    // main() 関数はこのまま使う. 修正しないこと!
{
    int Nx, Ny;
    printf("画像の横方向ピクセル数は? "); scanf("%d", &Nx);
    printf("画像の縦方向ピクセル数は? "); scanf("%d", &Ny);
    unsigned char* data = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
    if (data == NULL) // メモリ割当成功か?
    {
        printf("メモリエラー!!"); exit(0); } // メモリ割当に失敗したらプログラムを終了する
    Image img = { (unsigned char*) data, Nx, Ny };
    CglSetAll(img, 0);
    PaintCircle(img, Nx/2, Ny/2, Nx/5, 255);
    PaintCircle(img, 0, 0, Nx/8, 150);
    PaintCircle(img, Nx/2, Ny/8, Nx/6, 100);
    PaintCircle(img, 5*Nx/4, 3*Ny/4, Nx/2, 50);
    PaintCircle(img, Nx/2, 7*Ny/4, Nx, 200);
    CglSaveGrayBMP(img, "Circles.bmp");
    free(data); // メモリ解放
}
```

基本課題4の実行例

注) 文字「Sample」は描画不要

画像の横方向ピクセル数は？ 400
画像の縦方向ピクセル数は？ 400

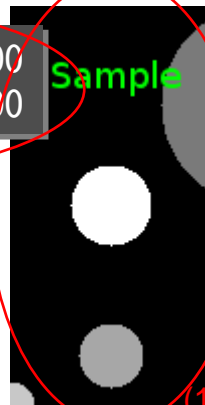
画像の横方向ピクセル数は？ 300
画像の縦方向ピクセル数は？ 200



画像の横方向ピクセル数は？ 100
画像の縦方向ピクセル数は？ 200

Sample

(2) 実行結果 (画面コピー)

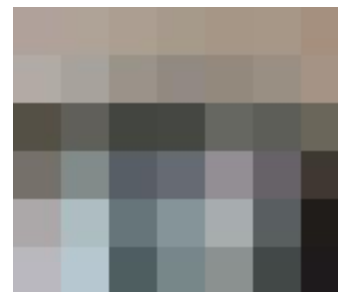
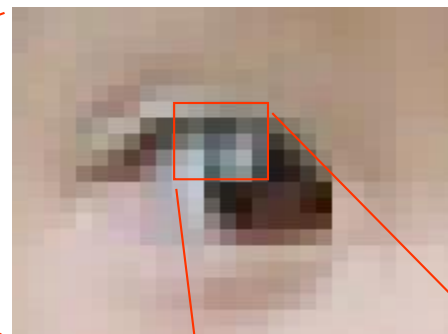


(1) 実行結果(BMPファイル)

- ・ ソースを提出
- ・ (1)と(2)の両方を提出！
- ・ 実行例を2種類以上提出！

カラーデジタル画像

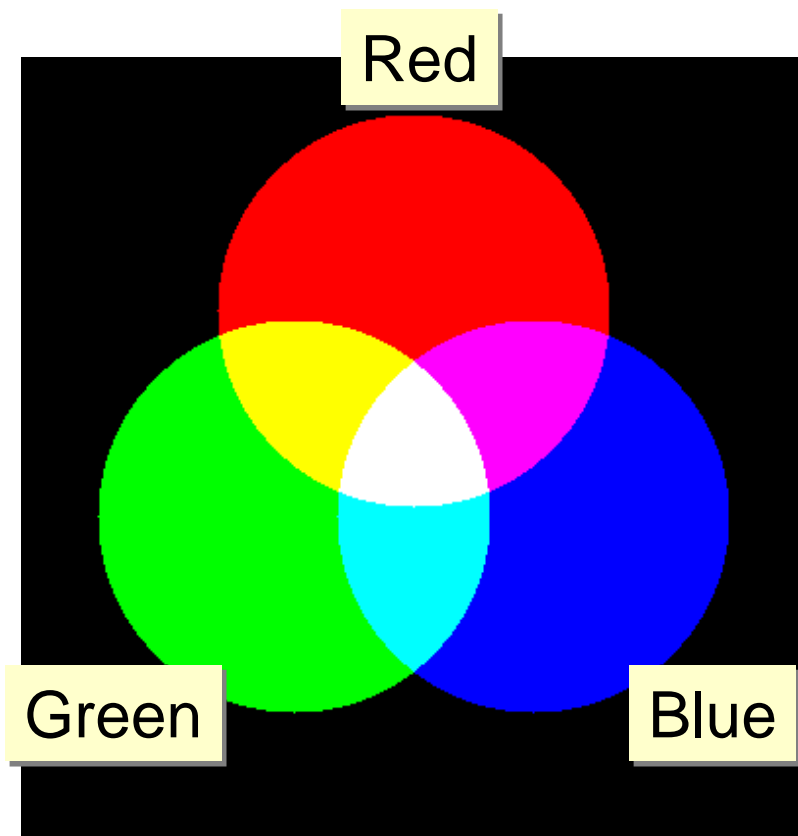
512 ピクセル



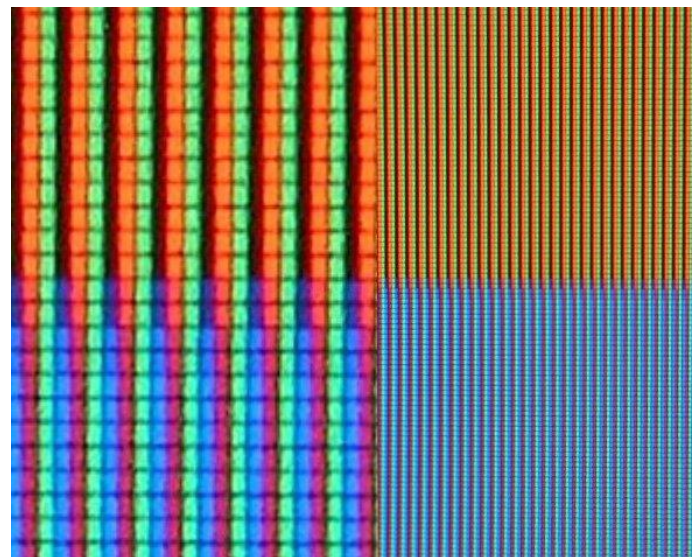
512 ピクセル

それぞれのピクセルが色を持っている

光の三原色とRGBカラーモデル



加法混色の例



カラー液晶の拡大写真*

* Wikipediaからの引用

コンピュータにおけるカラー表現

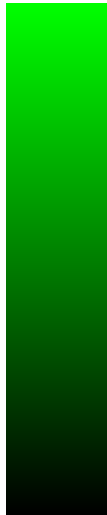


光のスペクトル*

* Wikipediaからの引用



Red



Green



Blue

255

⋮
⋮
⋮
⋮

0

0, 1, ..., 255: 明度

256階調

RGB各色が8ビット階調の明度を持ち、その組み合わせでカラーを表現
→ 1ピクセルの表現に3バイト必要

(R, G, B)

= (0, 255, 0)

→ 緑

= (255, 255, 255)

→ 白

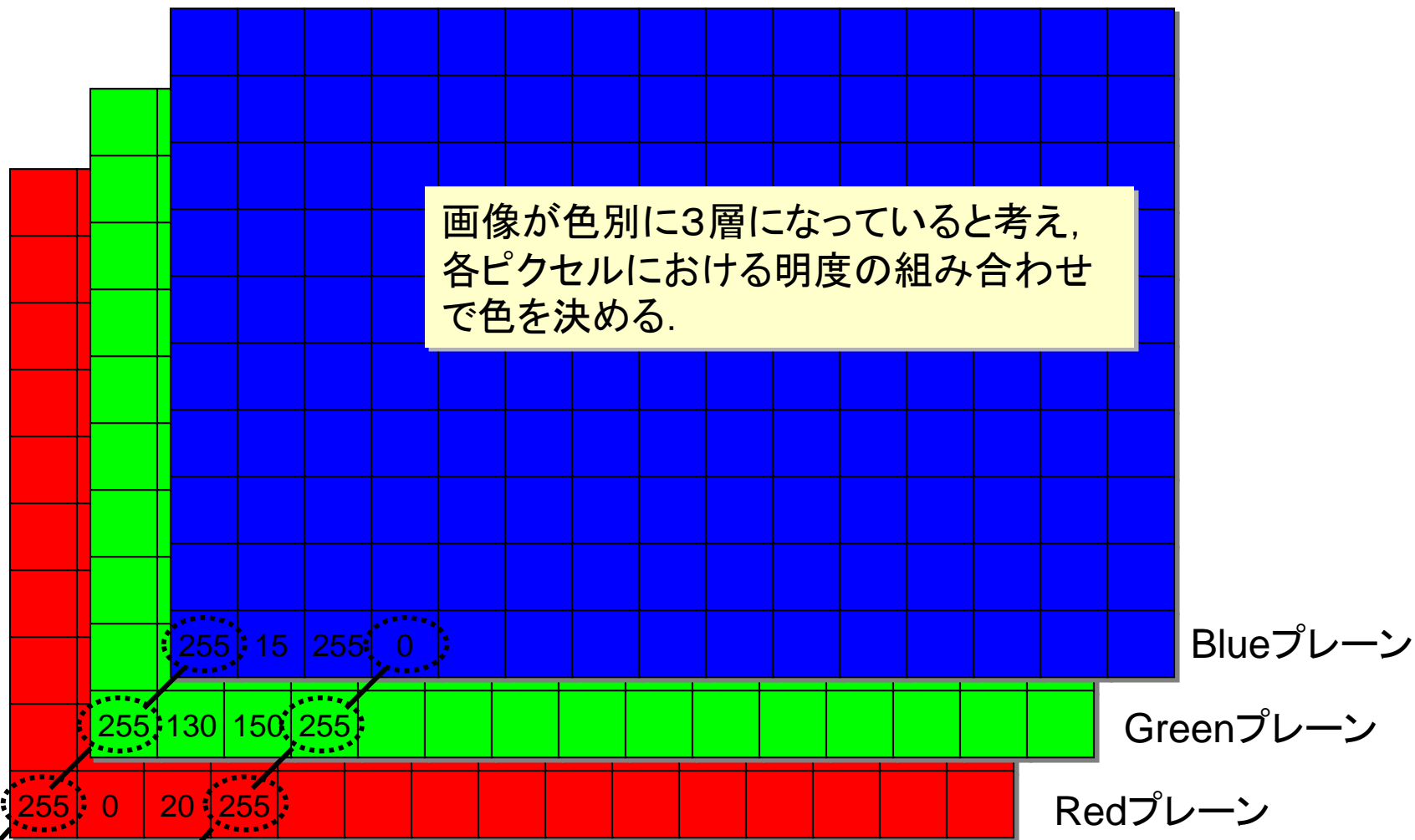
= (255, 255, 0)

→ 黄

これ以外にも、コンピュータでカラー画像を現す方式は種々あるが、
ここでは深く立ち入らない

プログラムでのカラー画像の取り扱い

画像が色別に3層になっていると考え、
各ピクセルにおける明度の組み合わせ
で色を決める。



(255, 255, 0) → 黄色

(255, 255, 255) → 白色

cglecライブラリの関数

```
void CglSaveColorBMP(Image red, Image green, Image blue,  
                     const char* fname)
```

カラー画像をBMPファイルとして保存する関数

red	赤色プレーンの画像を示すImage型構造体変数
green	緑色プレーンの画像を示すImage型構造体変数
blue	青色プレーンの画像を示すImage型構造体変数
fname	ファイル名(拡張子は「.bmp」にしておくこと)

カラー画像作成プログラムのソース例

Example4-3

```
#include "cglec.h"
#define WIDTH 256
#define HEIGHT 256

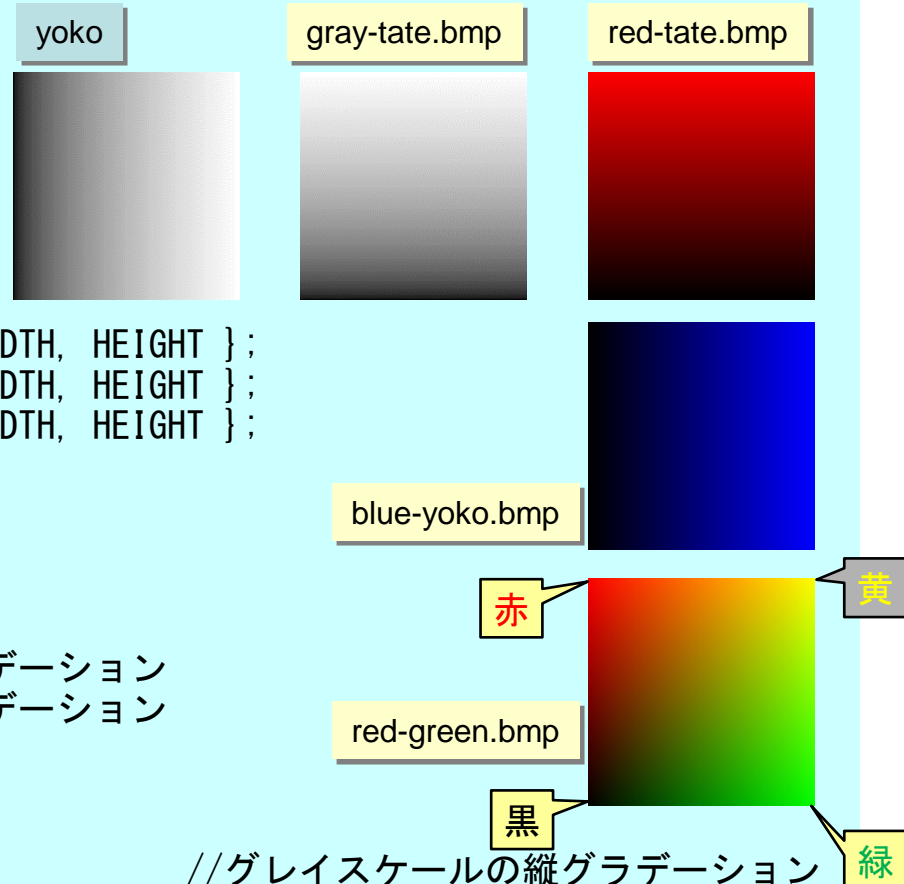
int main(void)
{
    unsigned char tate[WIDTH][HEIGHT];
    unsigned char yoko[WIDTH][HEIGHT];
    unsigned char kuro[WIDTH][HEIGHT];
    Image img_tate = { (unsigned char*) tate, WIDTH, HEIGHT };
    Image img_yoko = { (unsigned char*) yoko, WIDTH, HEIGHT };
    Image img_kuro = { (unsigned char*) kuro, WIDTH, HEIGHT };
```

```
    int i, j;
    for (i = 0; i < WIDTH; i++)
        for (j = 0; j < HEIGHT; j++)
        {
            tate[i][j] = j;    // 縦方向のグラデーション
            yoko[i][j] = i;    // 横方向のグラデーション
        }
```

```
    CglSetAll(img_kuro, 0);
```

```
    CglSaveGrayBMP(img_tate, "gray-tate.bmp");
    CglSaveColorBMP(img_tate, img_kuro, img_kuro, "red-tate.bmp"); // グレイスケールの縦グラデーション
    CglSaveColorBMP(img_kuro, img_kuro, img_yoko, "blue-yoko.bmp"); // 青色の横グラデーション
    CglSaveColorBMP(img_tate, img_yoko, img_kuro, "red-green.bmp"); // 赤と緑の縦横グラデーション
```

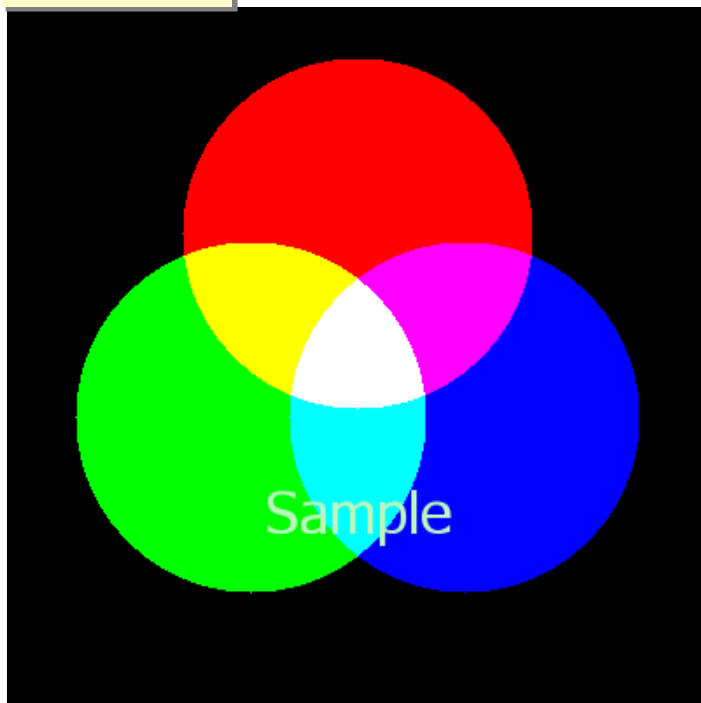
↑ Red ↑ Green ↑ Blue



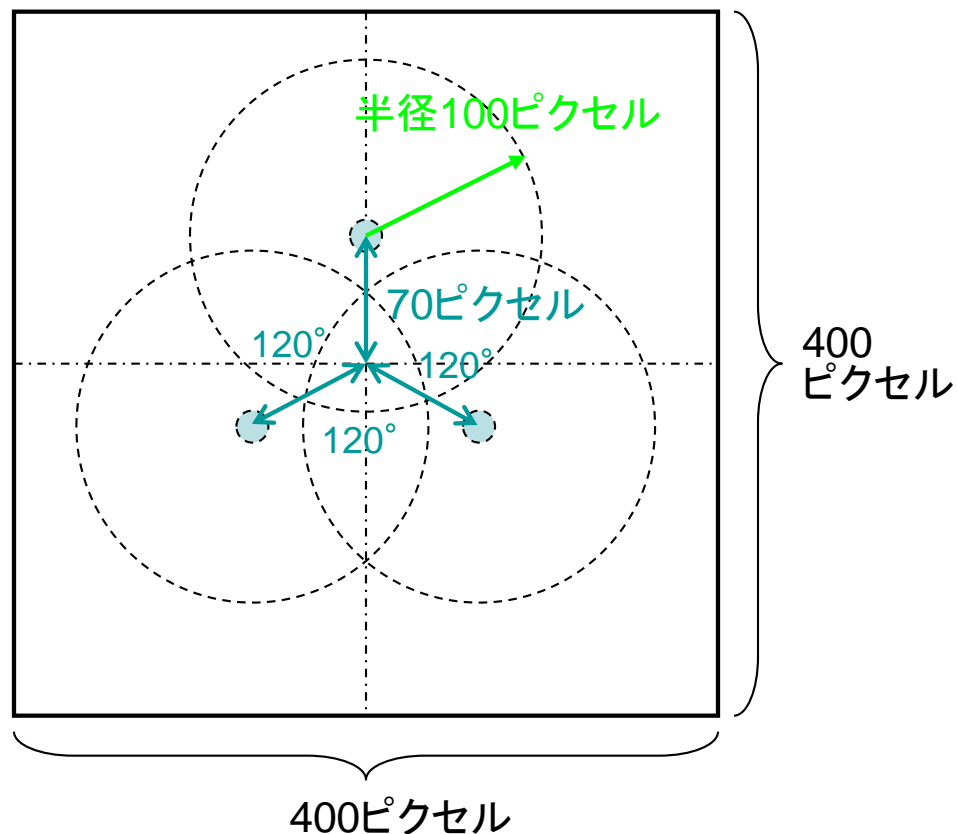
発展課題4

基本課題4で作成したPaintCircle()関数を再び用いて、下記の実行結果と同じ加法混色を示すカラー画像を作成するプログラムを作成しなさい。

実行結果



注) 図中の「Sample」の文字は、解答には必要ない



ソースと実行結果を提出