基本課題1 さらに良い解答

復習

正解例1

ループ回数：21 × 21 = 441 回

```
for (i = 0; i < WIDTH; i++)          // ゼロにクリアする
{
    for (j = 0; j < HEIGHT; j++)
        image[i][j] = 0;
}

for (i = 0; i < WIDTH; i++)  // 2重ループで縦横の座標値が等しければ1にする.
{
    for (j = 0; j < HEIGHT; j++)
        if (i == j)
            image[i][j] = 1;
}
```

合計
441+441=882

正解例3

ループ回数：21 × 21 = 441 回

```
for (i = 0; i < WIDTH; i++)
{
    for (j = 0; j < HEIGHT; j++)
        if (i == j)
            image[i][j] = 1;    //座標値が同じなら1
        else
            image[i][j] = 0;    //座標値が異なるなら0
}
```

合計
441

良くできました!

基本課題1 良くない例(≒不正解)

復習

```
for (i = 0; i < WIDTH; i++) // ゼロにクリアする
{
    for (j = 0; j < HEIGHT; j++)
        image[i][j] = 0;
}

for (i = 0; i < WIDTH; i++)
{
    image[i][i] = 1;
}

for (j = 0; j < HEIGHT; j++)
{
    image[j][j] = 1;
}
```

ムダ！

```
for (i = 0; i < WIDTH; i++) // ゼロにクリアする
{
    for (j = 0; j < HEIGHT; j++)
        image[i][j] = 0;
}

for (i = 0; i < WIDTH; i++)
{
    for (j = 0; j < HEIGHT; j++)
        image[i][i] = 1;
}
```

ムダ！

```
for (i = 0; i < WIDTH; i++) // ゼロにクリアする
{
    for (j = 0; j < HEIGHT; j++)
        image[i][j] = 0;
}

j = 0;
for (i = 0; i < WIDTH; i++)
{
    image[i][j] = 1;
    j++;
}
```

危険な不正アクセス！

復習

```
for (i = 0; i < WIDTH; i++)
{
    for (j = 0; j < HEIGHT; j++)
        image[i][j] = 0;
}

for (i = 0; i < WIDTH; i++)
{
    image[i][i] = 1;
}

for (j = 0; j < HEIGHT; j++)
{
    image[WIDTH][j] = 1;
}
```

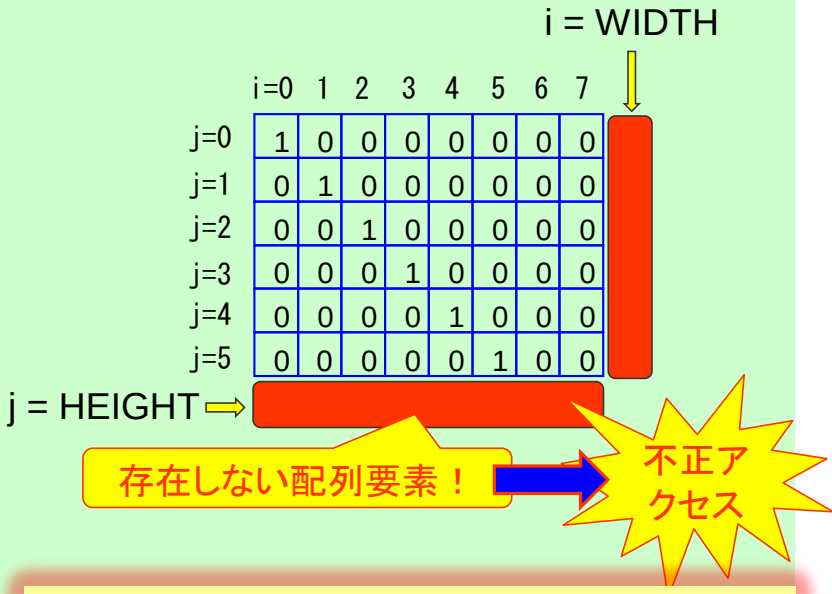
```
for (i = 0; i < WIDTH; i++) // ゼロにクリアする
{
    for (j = 0; j < HEIGHT; j++)
        image[i][j] = 0;
}

for (i = -1; i < WIDTH; i++)
{
    image[i][j] = 1;
    j++;
}
```

j=0に戻していない
ループ開始時 j=21

i=-1からループスタート??!

実行結果が正解と一致するのは単なる偶然！

```
#define WIDTH   8  // 画像の幅(ピクセル数)
#define HEIGHT  6  // 画像の高さ(ピクセル数)
配列 image[WIDTH][HEIGHT]
```



運が良い時 ⇒ 何も問題は起きない
(ただし正解と一致したのは偶然)

問題が起きるときの症状
- ✓ 途中でプログラムが停止しているが，一見正常終了
- ✓ 不正アクセス等のエラーメッセージで終了
- ✓ 突然，コンソールウィンドウ(黒ウィンドウ)が閉じる

今後はデバッガを利用せずに課題に解答するのは困難！

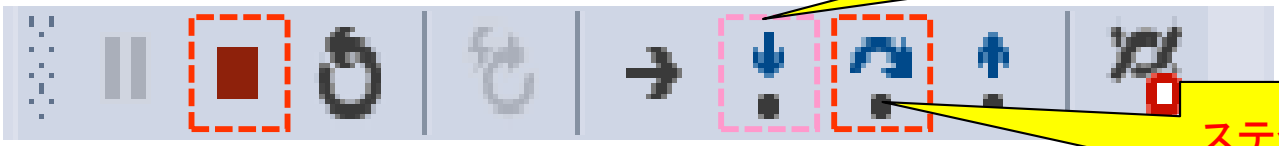
デバッガの利用

復習

ツールバーでデバッグ
(1)　ツールボタンで右クリックして，デバッグを選ぶ
(2)　デバッグツールバーからボタンを選ぶ

特定の行からデバッグを始める
ソースプログラムの特定の行をポイントしておいて，
右ボタンで「カーソル行の前まで実行」メニューを用いる

(a)ステップオーバー	次の行を実行
(b) ステップイン	次の行を実行
(c) デバッグの中止	デバッグをやめる



(c)　(b)　(a)

ステップインは関数内部に入って1行づつ実行する

ステップオーバーは関数内部に入らずに1行づつ実行する

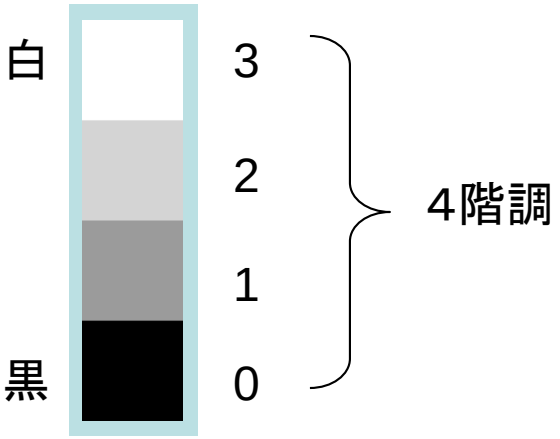
グレースケール画像

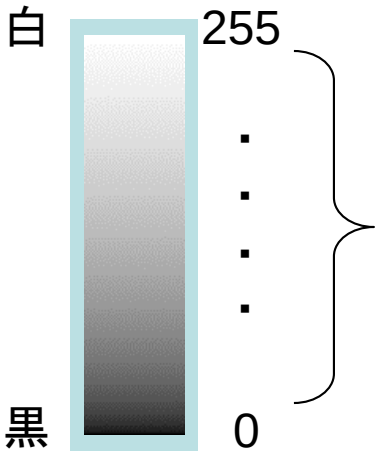
復習

カラー画像

グレースケール画像

階調 ＝ 1つのピクセルの明るさの段階





0, 1, ・・・, 255
階調値, ピクセル値, グレーレベル

1ピクセルが1バイトで表現できる

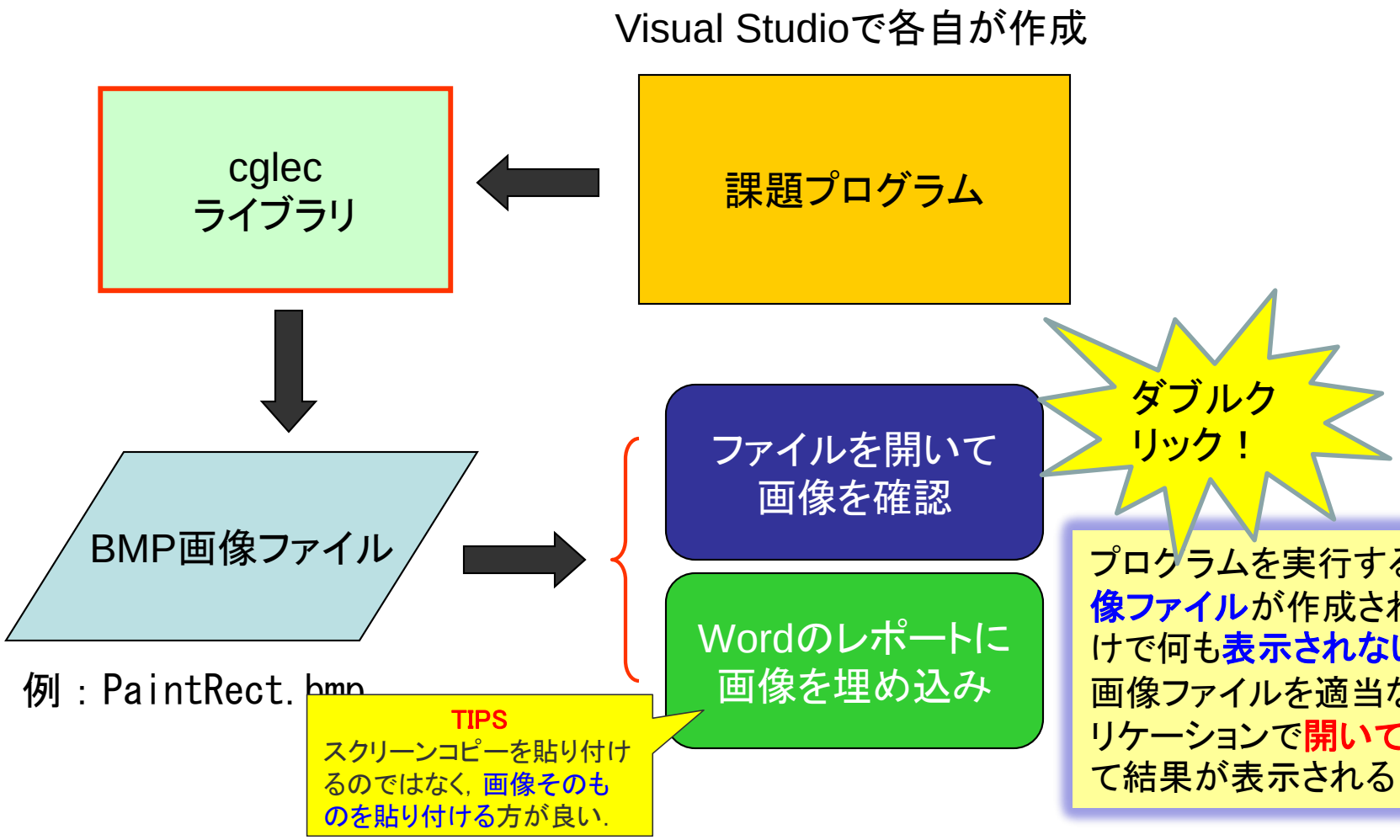
256=2^8 → 8ビットグレースケール

「深さ8ビットのグレースケール」と表現する場合もある

グレイスケール画像を描くCGプログラミング

復習

cglecライブラリ　：　この授業用に開発された2次元CG用ライブラリ

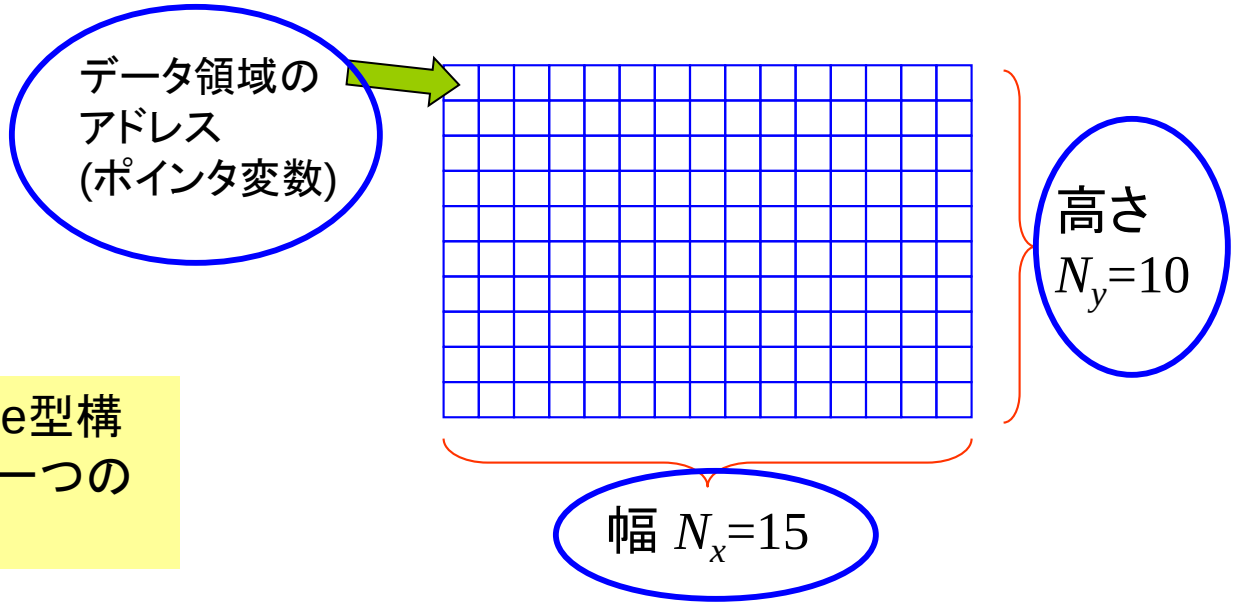


画像を管理する構造体の定義

復習

```
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
};
```

この構造体はヘッダファイルcglec.hの中で定義されている



一つのImage型構造体変数が一つの画像を表す

複数の画像を用いる時に便利

cglecライブラリを使って 8ビットグレースケール画像を描く

復習

Example2-1

```
#include    "cglec.h"
#define WIDTH   400
#define HEIGHT  400

int main(void)
{
    unsigned char data[WIDTH][HEIGHT];

    Image img = { (unsigned char*) data, WIDTH, HEIGHT };

    CglSetAll(img, 0);                  // imgをグレイレベル0でクリアする
    int x, y;

    for (y = 100; y < 150; y++)
    {
        for (x = 100; x < 200; x++)
            data[x][y] = 255;           // グレイレベル255(白)の四角形描画
    }

    for (y = 200; y < 250; y++)
    {
        for (x = 300; x < 350; x++)
            data[x][y] = 127;           // グレイレベル127(灰色)の四角形描画
    }

    CglSaveGrayBMP(img, "PaintRect.bmp");
}
```

cglecライブラリのヘッダファイル読み込み
(Image構造体の定義もここにある)

データ領域の確保

Image型構造体変数のimgを宣言して初期化

cglecライブラリのCglSetAll()関数を呼び出す

cglecライブラリのCglSaveGrayBMP()関数を呼び出す

cglecライブラリの関数

復習

```
void CglSetAll(Image img, unsigned char level)
```

画像全体を一つのグレーレベルで塗りつぶす関数
img　　塗りつぶす画像を示すImage型構造体変数
level　グレーレベル

```
void CglSaveGrayBMP(Image img, const char* fname)
```

画像をBMPファイルとして保存する関数
img　　保存する画像を示す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)

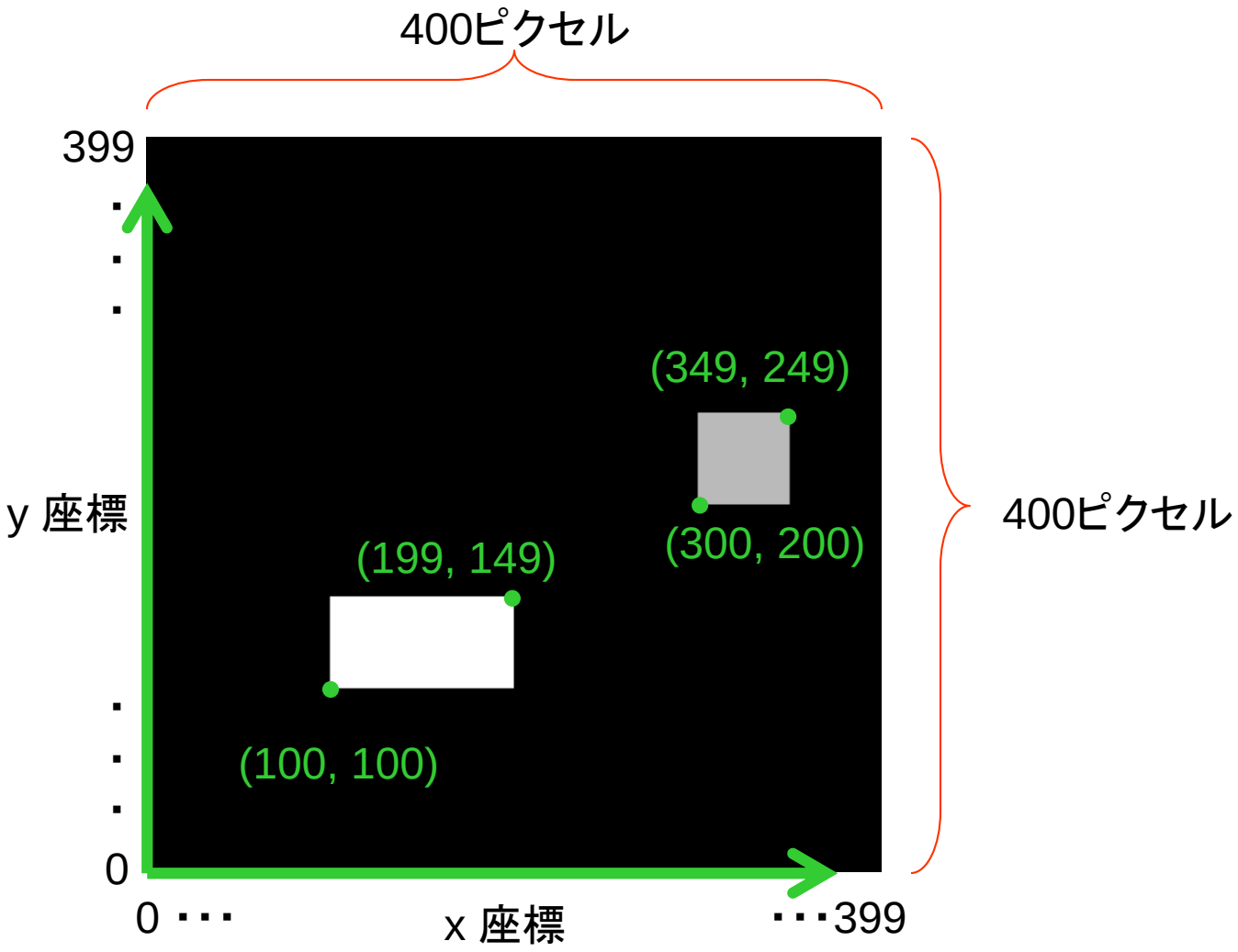
プロジェクトフォルダ
各自が作成したソースファイル(○○○.cpp)があるフォルダ

cglecライブラリを使うには
LMSから，ファイルcglec110.zipをダウンロードし，それを解凍して中にある次の3つのファイルをVisual Studioのプロジェクトフ
cglec.h
cglec.lib　⇒ この2つはコンパイル(ビル
cglec.dll　⇒ プログラム実行時に必要な
注) 新たにプロジェクトを作成するたびにそのプロ
ファイルをコピーしておく必要がある．(いちいちコピ
できるが，少し設定が難しい)

注意
cglecには，この他にも様々な関数が組み込まれていますが，課題の出題時点でまだ説明していない関数を使用して課題を解答しても0点です．

実行結果(PaintRect.bmpファイル)

復習

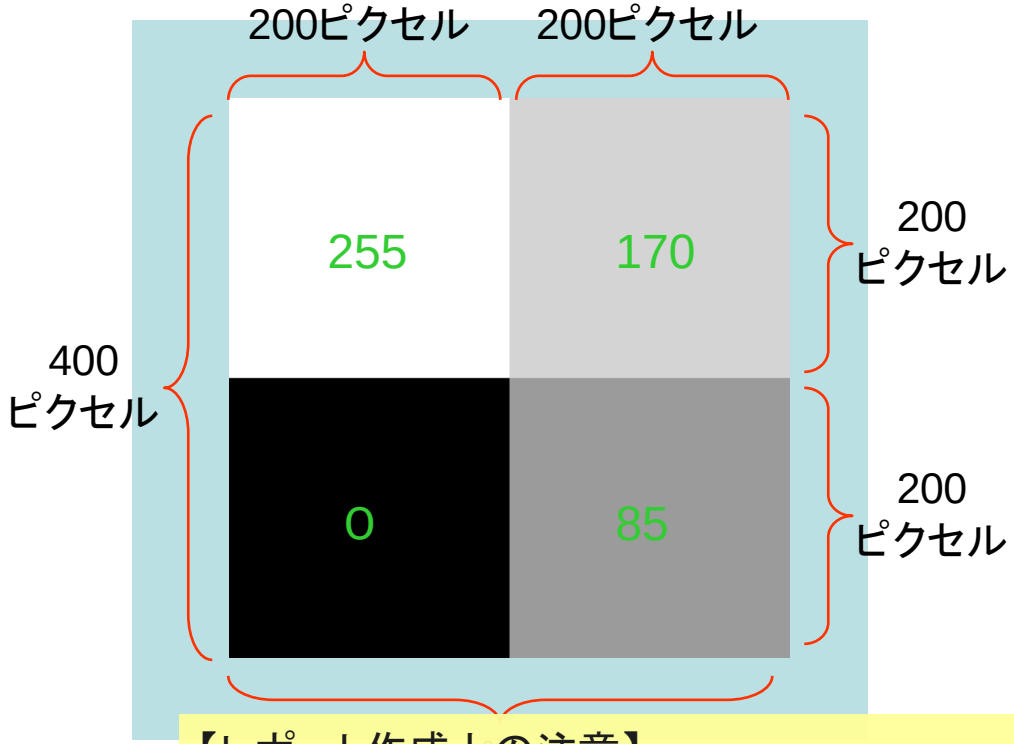


（注）必ず最新のcglec（cglec111.zip)を使用してください．Rel.1.1.0以前のcglecではガンマ値が1.0であるため，この結果と一致しません．

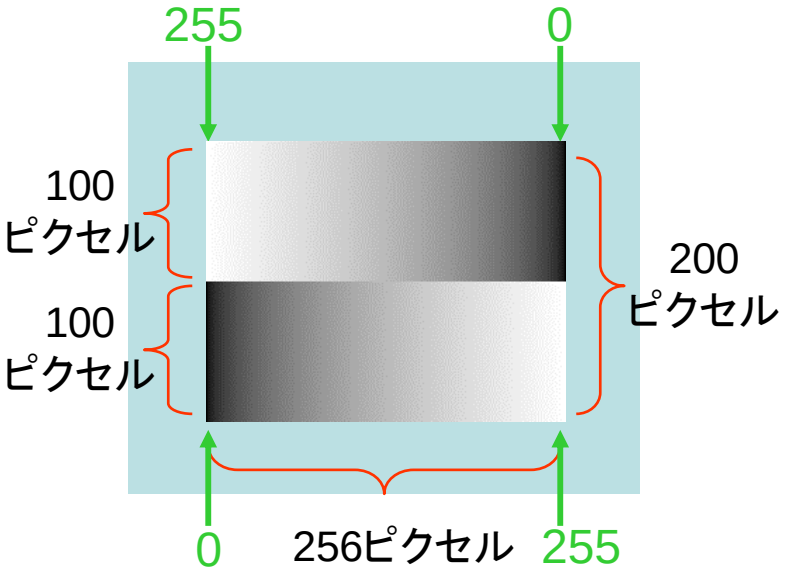
今日の課題

次のようなグレースケール画像を作り出すプログラムを作成せよ．
（緑色の数字はグレーレベル）

基本課題2　必ず提出



発展課題2　できれば提出



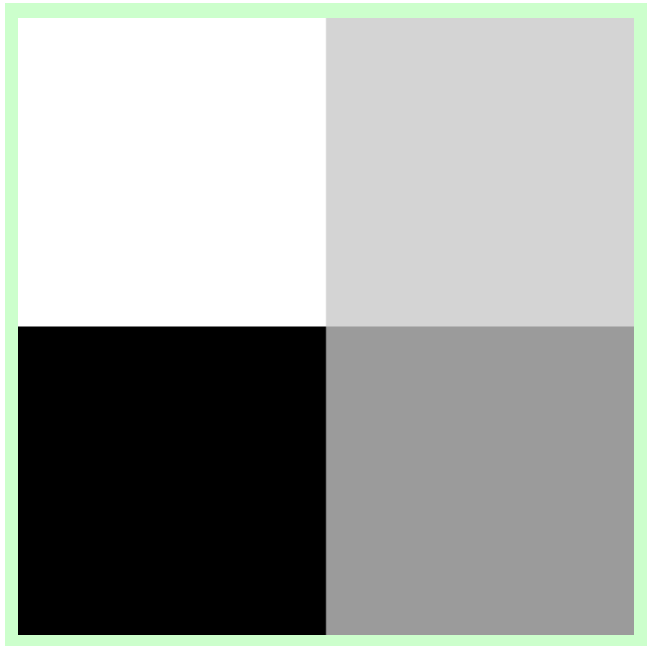
※階調が1レベルづつ連続的に変化すること

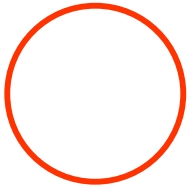
【レポート作成上の注意】
1. 画像ファイルを開いたスクリーンコピーをWord文書に貼り付けるのではなく，画像ファイルそのものを直接Word文書に貼り付けること
2. デバッグにはデバッガを活用すること

基本課題2 解答例

A君解答

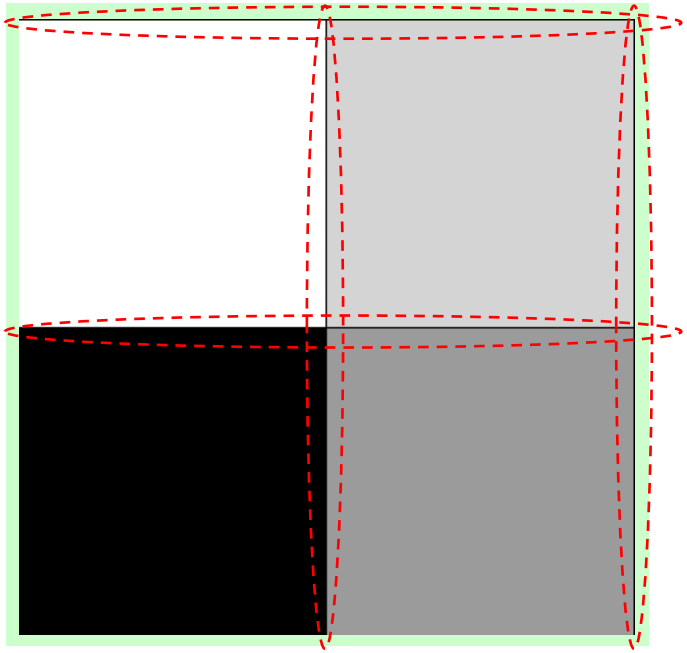
```
for(y=0;y<200;y++)
{
    for(x=200;x<400;x++)
        data[x][y]=85;
}
for(y=200;y<400;y++)
{
    for(x=0;x<200;x++)
        data[x][y]=255;
}
for(y=200;y<400;y++)
{
    for(x=200;x<400;x++)
        data[x][y]=170;
}
```





B君解答

```
for (y = 200; y < 399; y++)
{
    for (x = 0; x < 199; x++)
    data[x][y] = 255;
}
for (y = 200; y < 399; y++)
{
    for (x = 200; x < 399; x++)
        data[x][y] = 170;
}
for (y = 0; y < 199; y++)
{
    for (x = 200; x < 399; x++)
        data[x][y] = 85;
}
```



基本課題2 解答例

C君解答

```
for (y = 0; y < 400; y++){
    for (x = 0; x < 400; x++){
        if(y<200 && x<200){data[x][y] = 0;}
        if(y<200 && x>=200){data[x][y] = 85;}
        if(y>=200 && x<200){data[x][y] = 255;}
        if(y>=200 && x>=200){data[x][y] = 170;}
    }
}
```

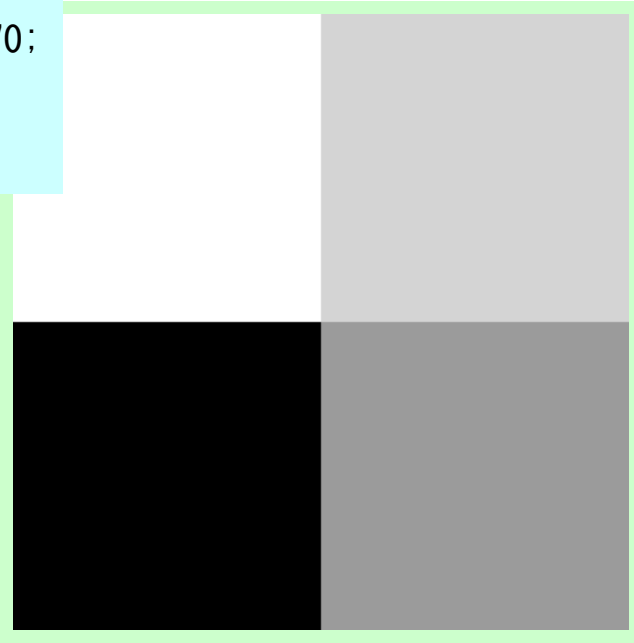
ループ回数
400 × 400 = 160,000 回

D君解答

```
for (y = 0; y <= HEIGHT / 2; y++)
    {
        for (x = 0; x <= WIDTH / 2; x++)
        {
            data[x][y] = 0;
            data[WIDTH / 2 + x][y] = 85;
            data[WIDTH / 2 + x][HEIGHT / 2 + y] = 170;
            data[x][HEIGHT / 2 + y] = 255;
        }
    }
```

ループ回数
200 × 200 = 40,000 回

良くできました!

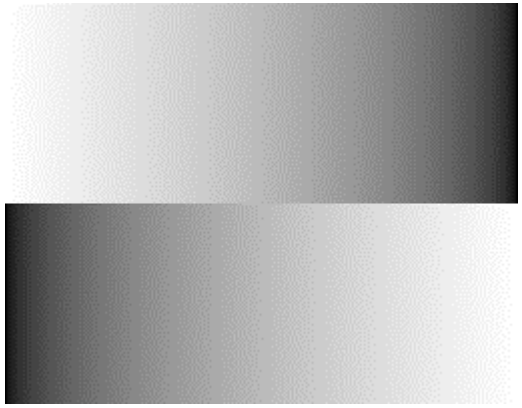


ループの中ではif～else等の条件分岐を可能な限り避ける
理由：現代のCPUは長いパイプラインを持つため，条件分岐が発生すると速度が落ちる

発展課題2 解答例

E君解答

```
#define WIDTH   256
#define HEIGHT  200
int main(void)
{
    unsigned char data[WIDTH][HEIGHT];
    Image img = { (unsigned char*) data, WIDTH, HEIGHT };
    SetAll(img, 0);
    int x, y;
    for(x=0;x<256;x++){
        for(y=0;y<100;y++){
            data[x][y] = x;
            data[x][y+100] = 255-x;
        }
    }
    SaveGrayBMP(img, "DrawRect.bmp");
}
```



F君解答

```
unsigned char data[WIDTH][HEIGHT];
Image img = { (unsigned char*) data, WIDTH, HEIGHT };
SetAll(img, 0);
int x, y;
for (y=0; y<100; y++)
{
    for (x=0; x<WIDTH; x++)
        data[x][y] = x;
}
for (y=100; y<200; y++)
{
    for (x=0; x<WIDTH; x++)
        data[x][y] = WIDTH-1-x;
}
SaveGrayBMP(img, "DrawRect.bmp");
```

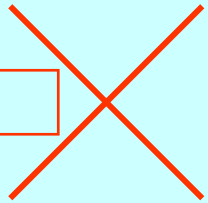
Good!

次にようにするとなお良い
200 → HEIGHT
100 → HEIGHT/2

100や256等の定数値は可能な限りソースプログラム中に書かない
理由：場合によってはこれらの値を変更したくなる時もあり，その場合すべての定数値を誤りなく書き換えなければならなくなるため．→できるだけマクロや変数を使うこと．

画像のプログラムでポインタが必要な理由

```
#include    <stdio.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[Nx][Ny];

    Image img = { (unsigned char*) data, Nx, Ny };
    CglSetAll(img, 0);
    // ・・・
    // 何かの処理
    // ・・・
    CglSaveGrayBMP(img, "Rei3-1.bmp");
}
```



メモリの静的割り当て

問題点：C言語の配列宣言では要素数は定数でなければならない
(コンパイルの時点で要素数が決まっていなければならない)

メモリの静的割り当て

```
unsigned char data[Nx][Ny];
```

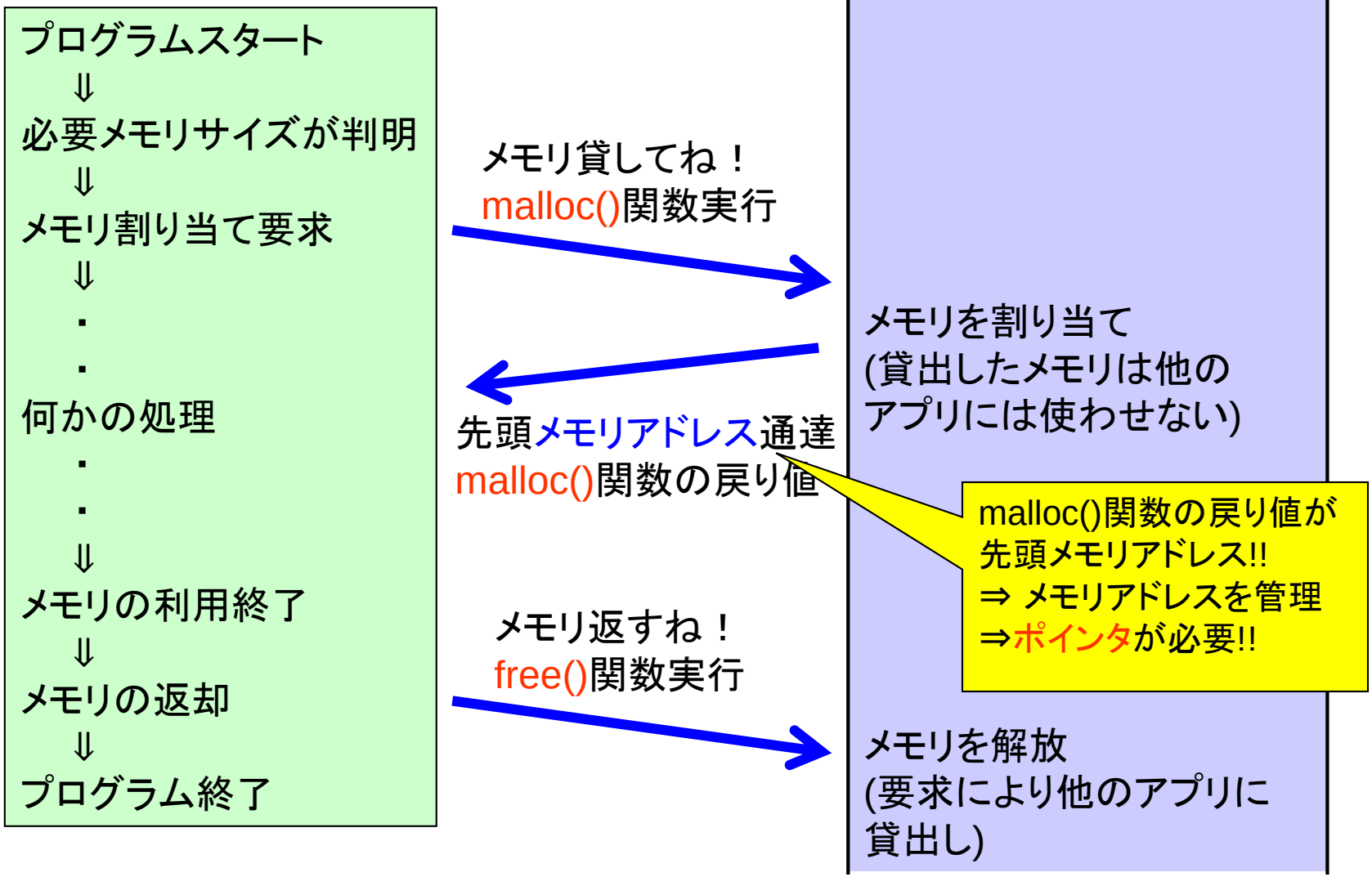
メモリの動的割り当て

```
unsigned char* data;
data = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
```

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)



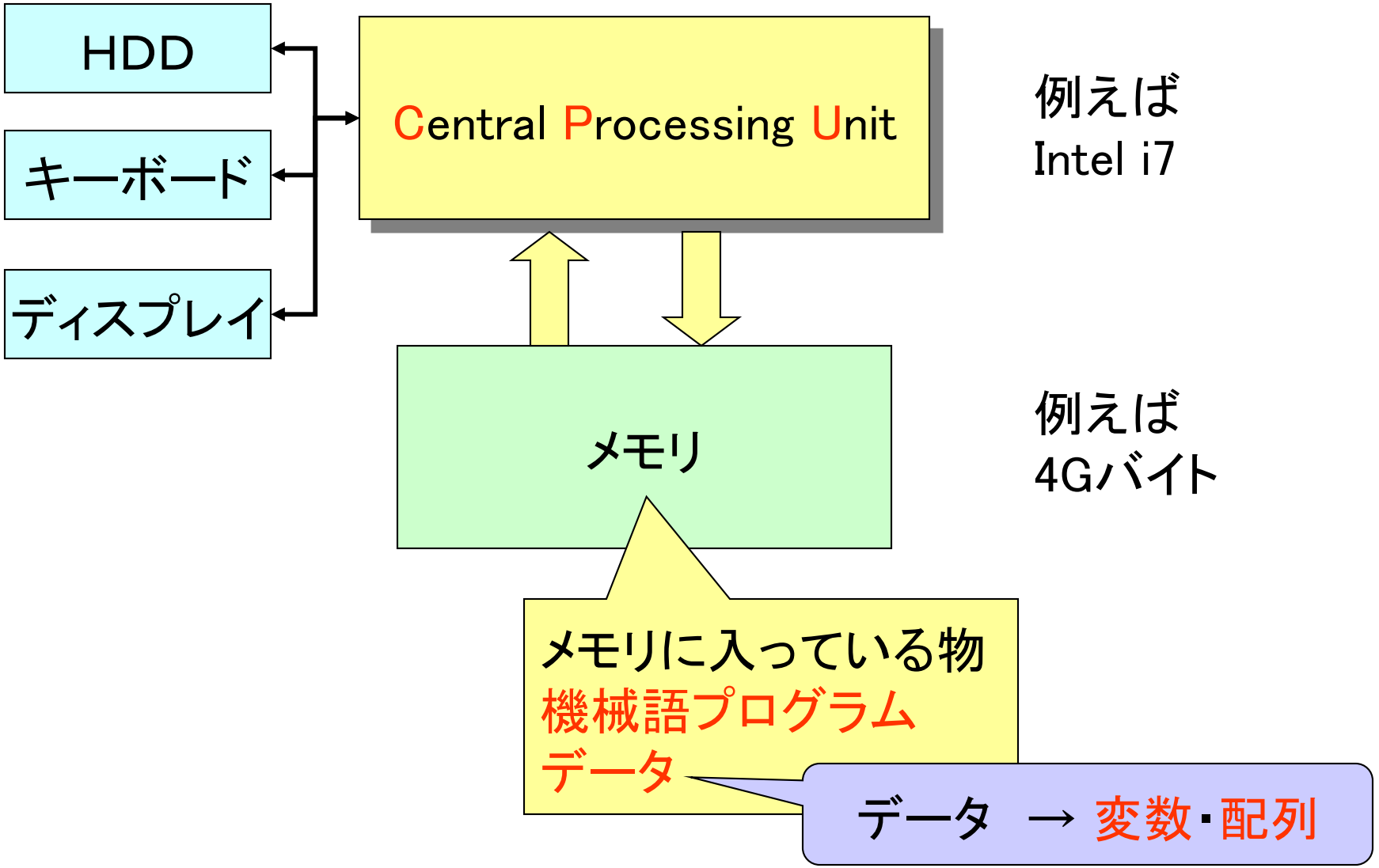
今度こそ理解するぞ，ポインタ！

- 現代のたいていのプログラミング言語にはポインタに類似した仕組みがある．
- 例えば，C#, Javaでは参照型変数
- C言語のポインタはそれらの中で，最も原始的で強力．
- ハードウェアを直接操作することに近いため，容易にメモリ内容を破壊する → 不正アクセス！

運が良い時 ⇒ 何も問題は起きない
(ただし正解と一致したのは偶然)

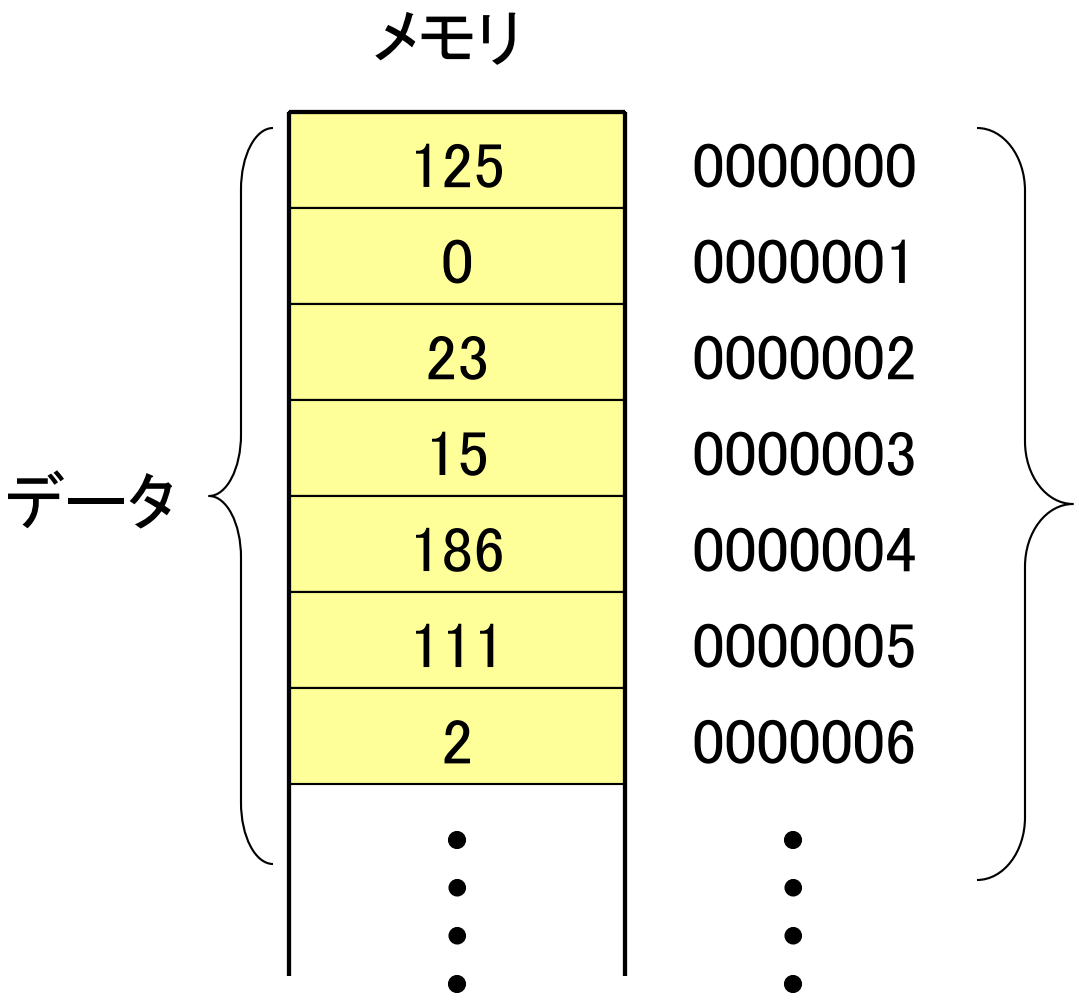
問題が起きるときの症状
✓ 途中でプログラムが停止しているが，一見正常終了
✓ 不正アクセス，例外などのエラーメッセージで終了
✓ 突然，コンソールウィンドウ(黒ウィンドウ)が閉じる

コンピュータの構造


HDD
キーボード
ディスプレイ
Central Processing Unit
例えば
Intel i7
メモリ
例えば
4Gバイト
メモリに入っている物
機械語プログラム
データ
データ → 変数・配列

メモリの構造

近年64ビットCPUも良く用いられるが，その場合はアドレスの長さは64ビット=8バイト

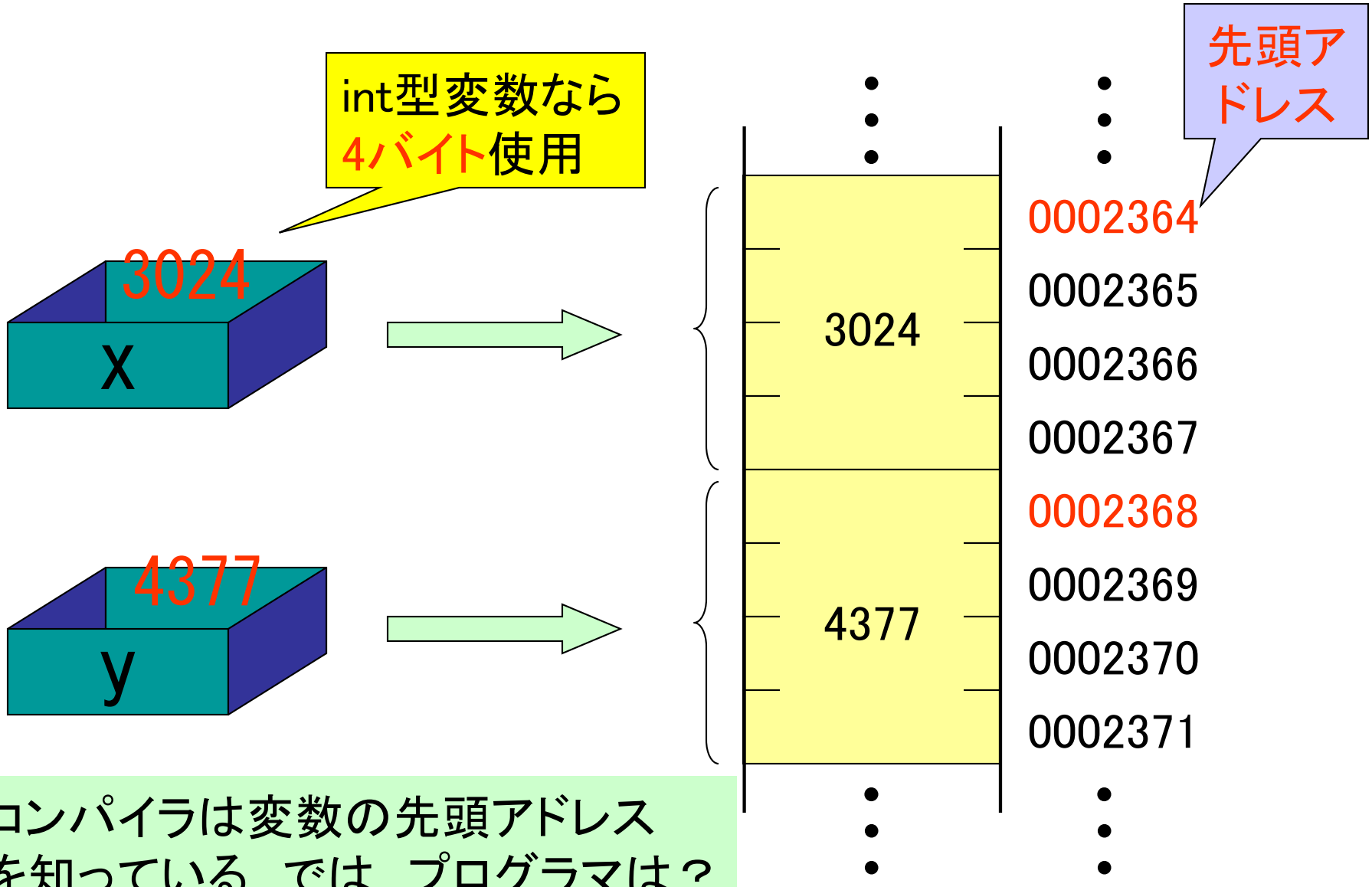


32ビットCPUなら，
アドレスの長さは
32ビット ＝ 4バイト

$2^{32} \fallingdotseq 4 \times 10^9$
約40億個の
アドレス

アドレスは
メモリ1バイト毎に
付けられている

変数はメモリにどのように格納されているか？



コンパイラは変数の先頭アドレス
を知っている．では，プログラマは？

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

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

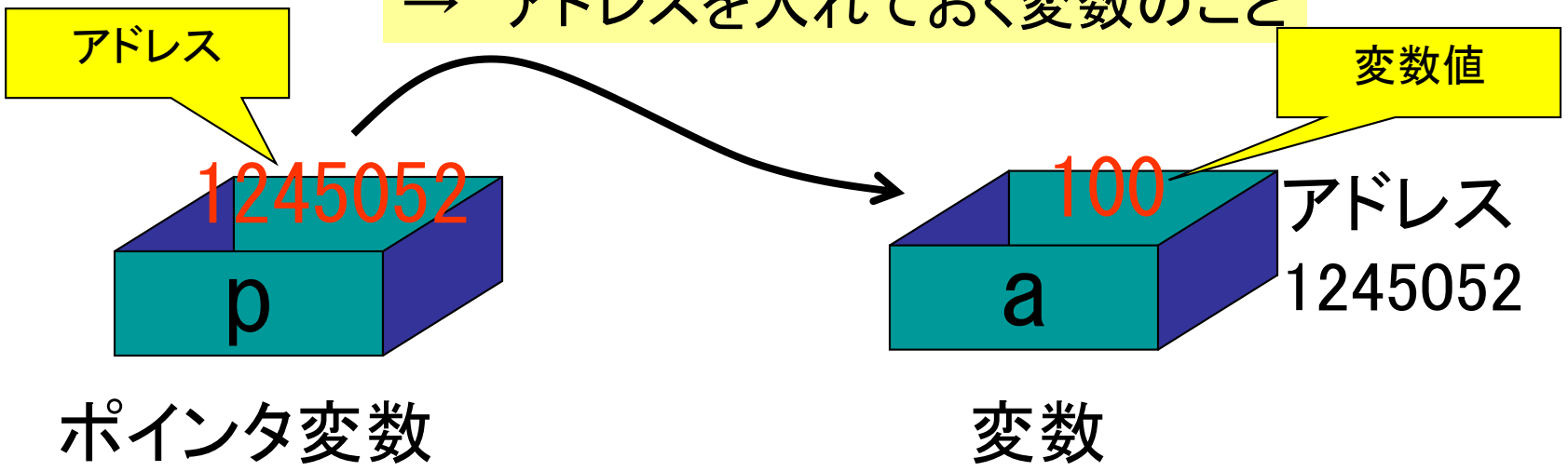
```
int a = 100;
int *p;
p = &a;
printf("値は%dで，アドレスは%dである¥n", a, p)
```

```
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(2)

ポインタ(ポインタ変数)とは？

→　アドレスを入れておく変数のこと



ポインタ変数は
別の変数を指し示している

（英単語）point
= ～を指差す，
～を示す

注意：ポインタ変数も変数の一種なので，当然メモリ内に格納され，
アドレスも持っている．ポインタ変数を指すポインタ変数もある．

変数のアドレスとポインタ(3)

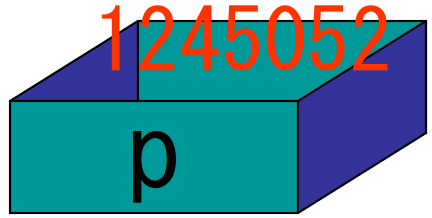
```
int a = 100;
int *p;
p = &a;
printf("値は%dで，アドレスは%dである¥n", *p, p)
```

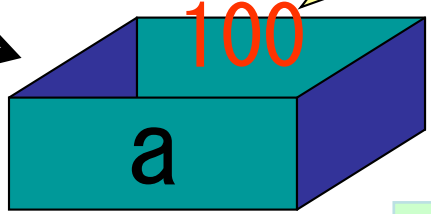
```
値は100で，アドレスは1245052である
```

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

*p の値は100





アドレス
1245052

ポインタ変数

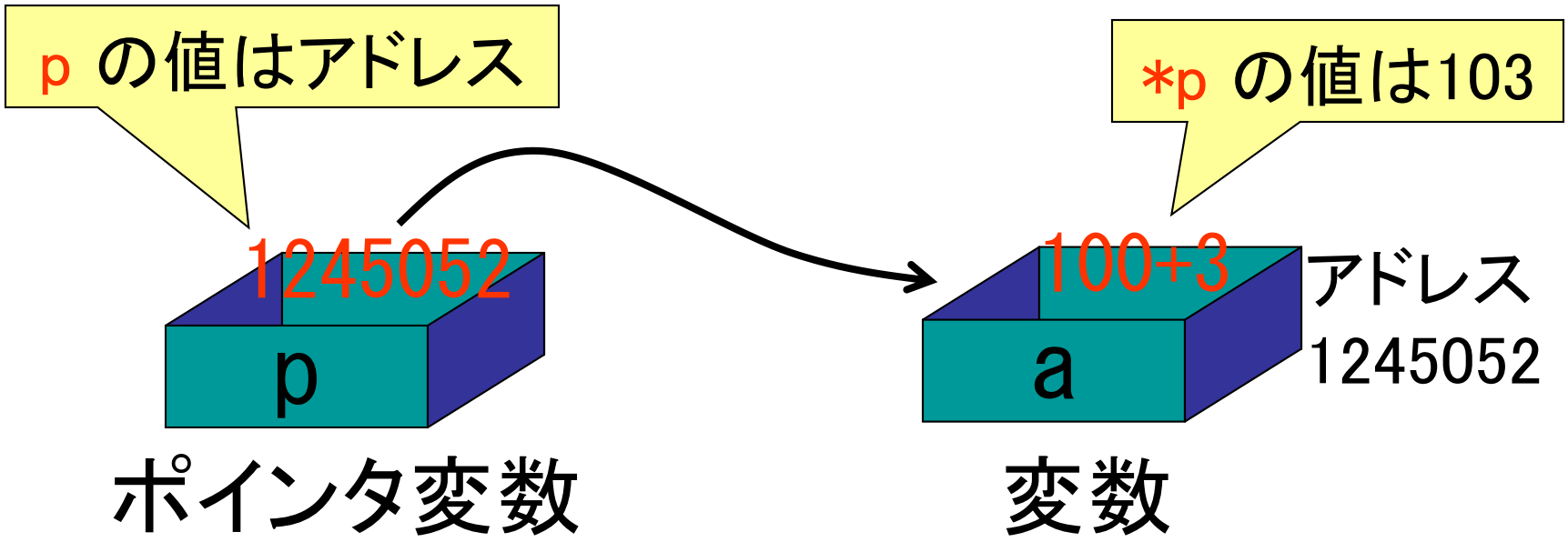
変数

＊p ⇔ a
同じ値

変数のアドレスとポインタ(4)

```
int a = 100;
int *p;
p = &a;
a = a + 3;
printf("aの値は%dで，*pの値は%dである¥n", a, *p)
```

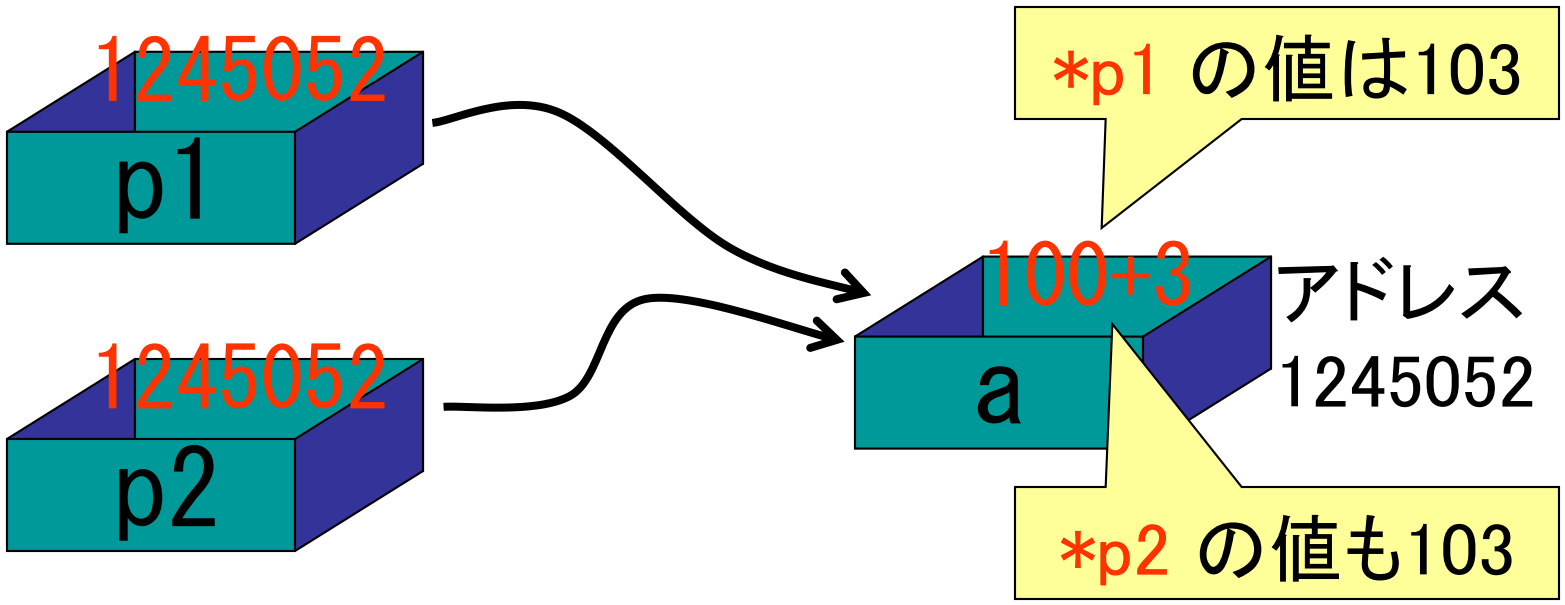
```
aの値は103で，*pの値は103である
```



変数のアドレスとポインタ(5)

```
int a = 100;
int *p1, *p2;
p1 = &a;
p2 = &a;
*p1 = *p1 + 3;
printf("*p1の値は%dで，*p2の値も%dである¥n", *p1, *p2)
```

*p1の値は103で，*p2の値も103である

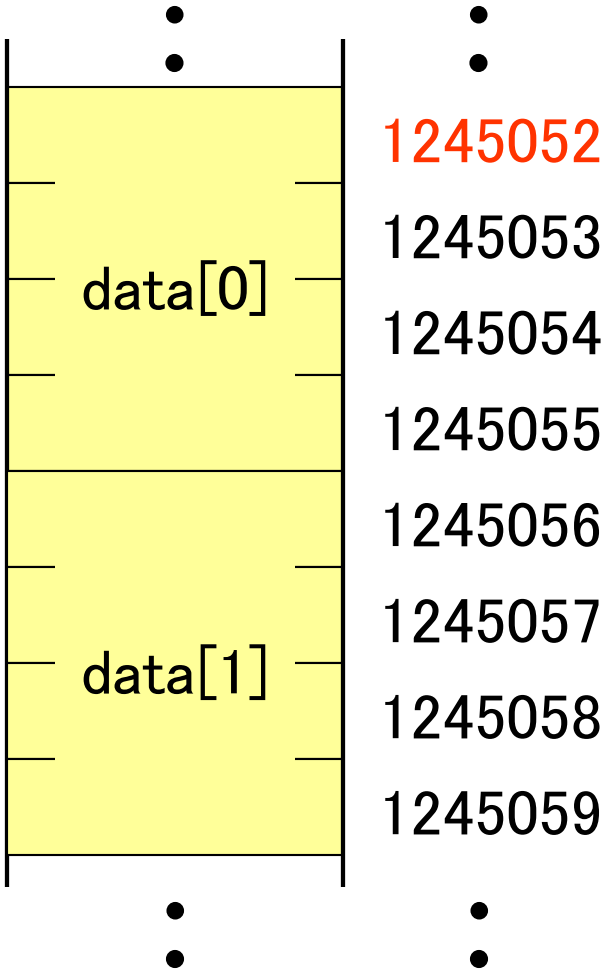


1次元配列のアドレスとポインタ

```
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

メモリの静的割り当てを用いたデータ領域の確保とImage構造体変数へのアドレスの設定

```
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
};
```

```
#include    "cglec.h"
#define WIDTH   400
#define HEIGHT  400

int main(void)
{
    unsigned char data[WIDTH][HEIGHT];

    Image img = { (unsigned char*) data, WIDTH, HEIGHT };

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

      .
      .
      .

    CglSaveGrayBMP(img, "PaintRect.bmp");
}
```

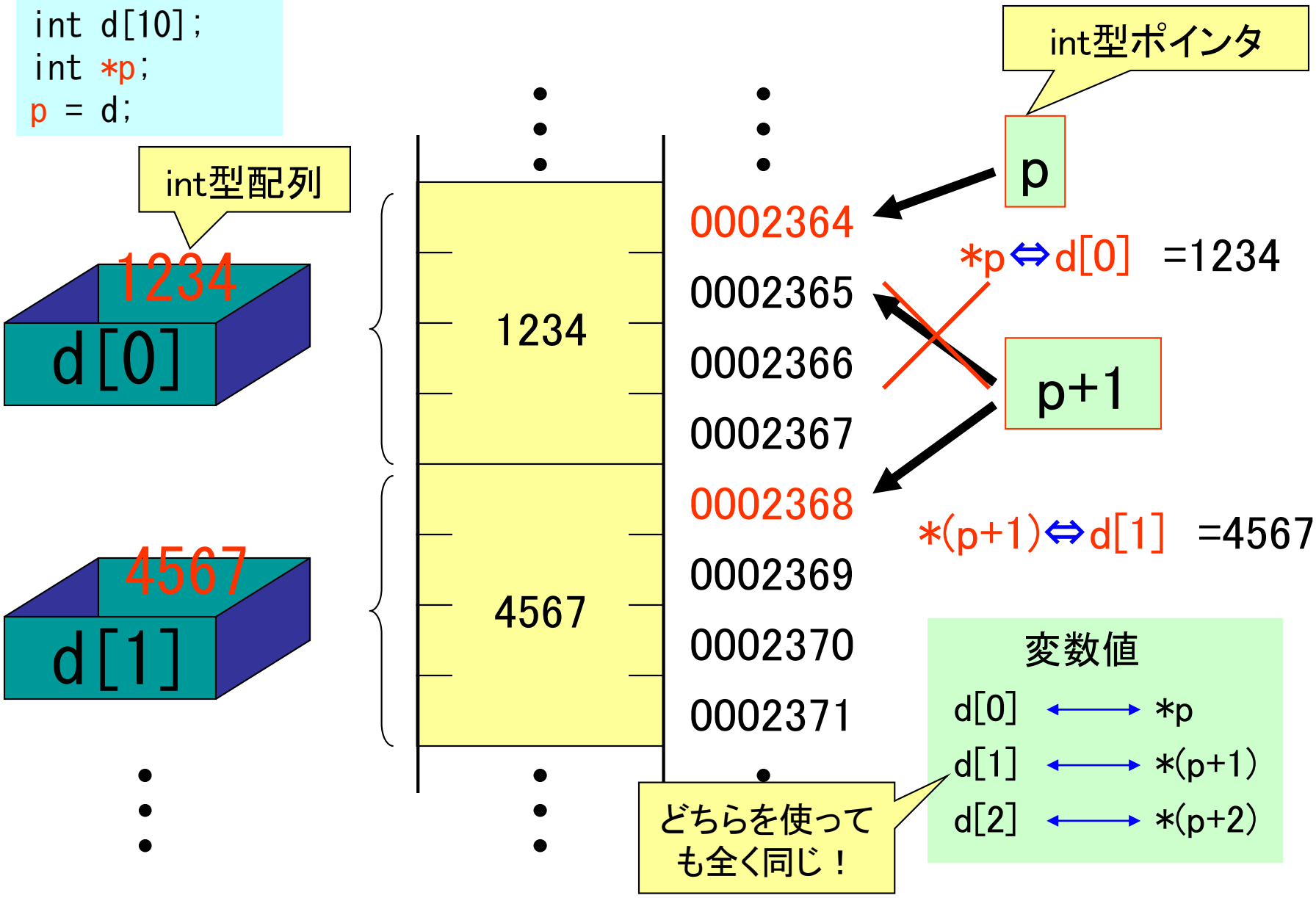
静的なメモリ割り当てを用いたデータ領域の確保

Image型構造体変数のimgを初期化するときにデータ領域のアドレスを設定
data → 配列の先頭アドレス

注)(unsigned char*)はunsigned char型のポインタへのキャスト(型変換)

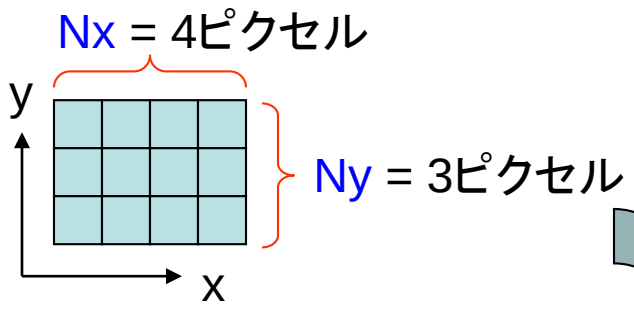
ポインタと1次元配列の関係

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



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・・

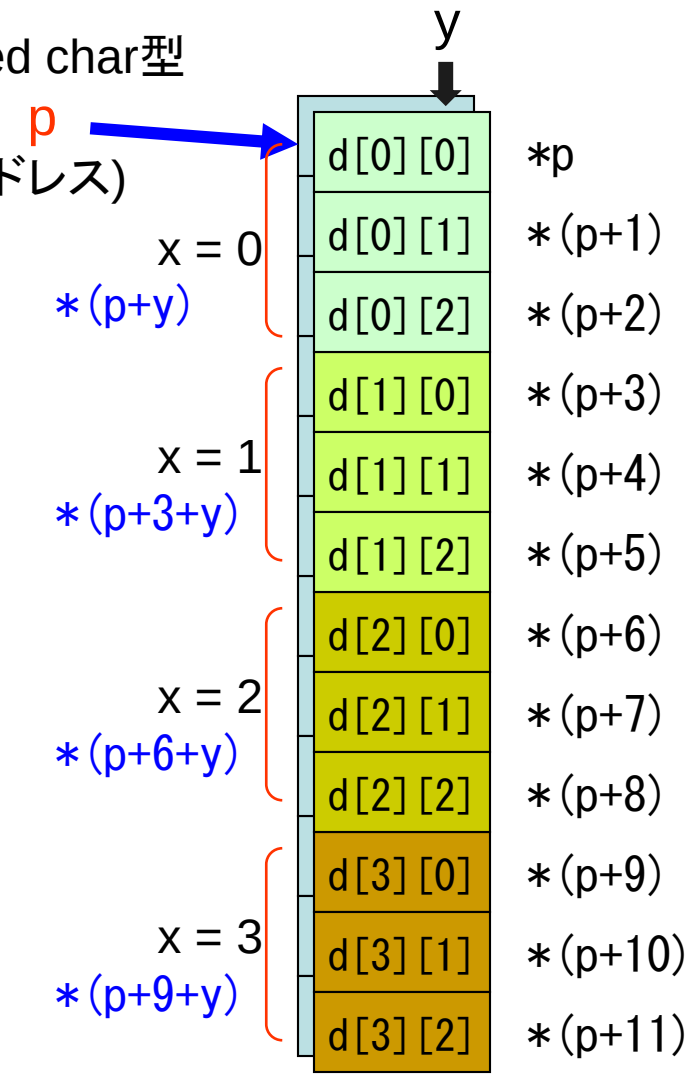


下の様な配列で考えると・・

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

	x = 0	x = 1	x = 2	x = 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]

unsigned char型
ポインタ p
(先頭アドレス)



d[x][y] ⟶ *(p + x*Ny + y)

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)                    //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);
    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);                          //メモリ解放
}
```

必要

unsigned char型のポインタ

unsigned char型ポインタへの変換

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; ×

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

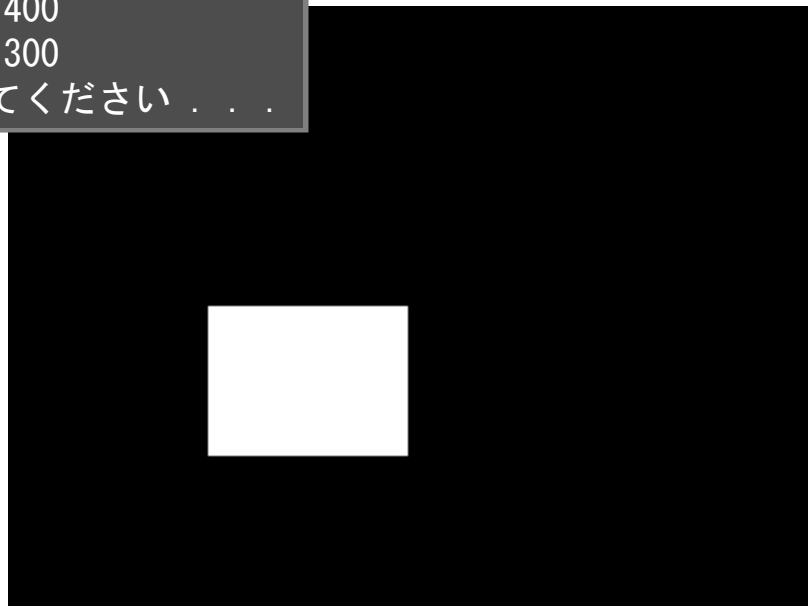
sizeof(「データ型」)
「データ型」の変数一つに必要なメモリのバイト数を返す演算子
注)
sizeof(unsgined char) ⇒ 1
従って簡易的に書くなら
・・・malloc(Nx*Ny)・・・
としても良い.

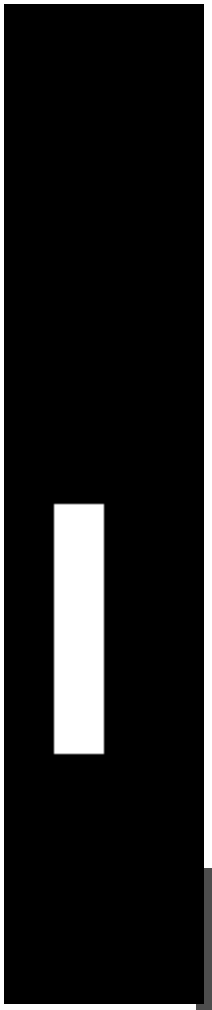
void free(void* mem)
ヘッダファイル stdlib.h
引数 mem：解放するメモリの先頭アドレス
戻り値 なし

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.

実行結果

```
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？ 300
続行するには何かキーを押してください . . .
```





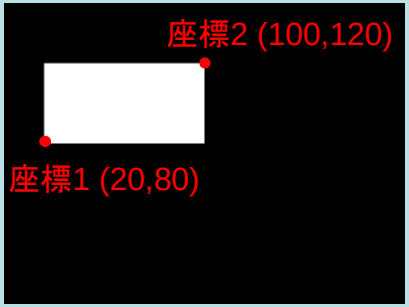
```
画像の横方向ピクセル数は？ 100
画像の縦方向ピクセル数は？ 30
続行するには何かキーを押してください . . .
```

```
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？ 500
続行するには何かキーを押してください . . .
```

基本課題3

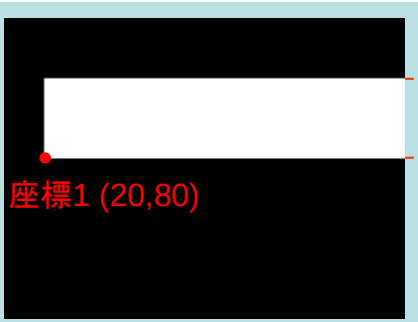
画像サイズを入力し，次に四角形の左下の座標(x_1, y_1)と右上の座標(x_2, y_2)を入力するとその四角形を白色で塗りつぶすプログラムを作成せよ．ただし(x_2, y_2)の位置が画像の外側である場合は画像の範囲内で四角形を描くこと．

```
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
```

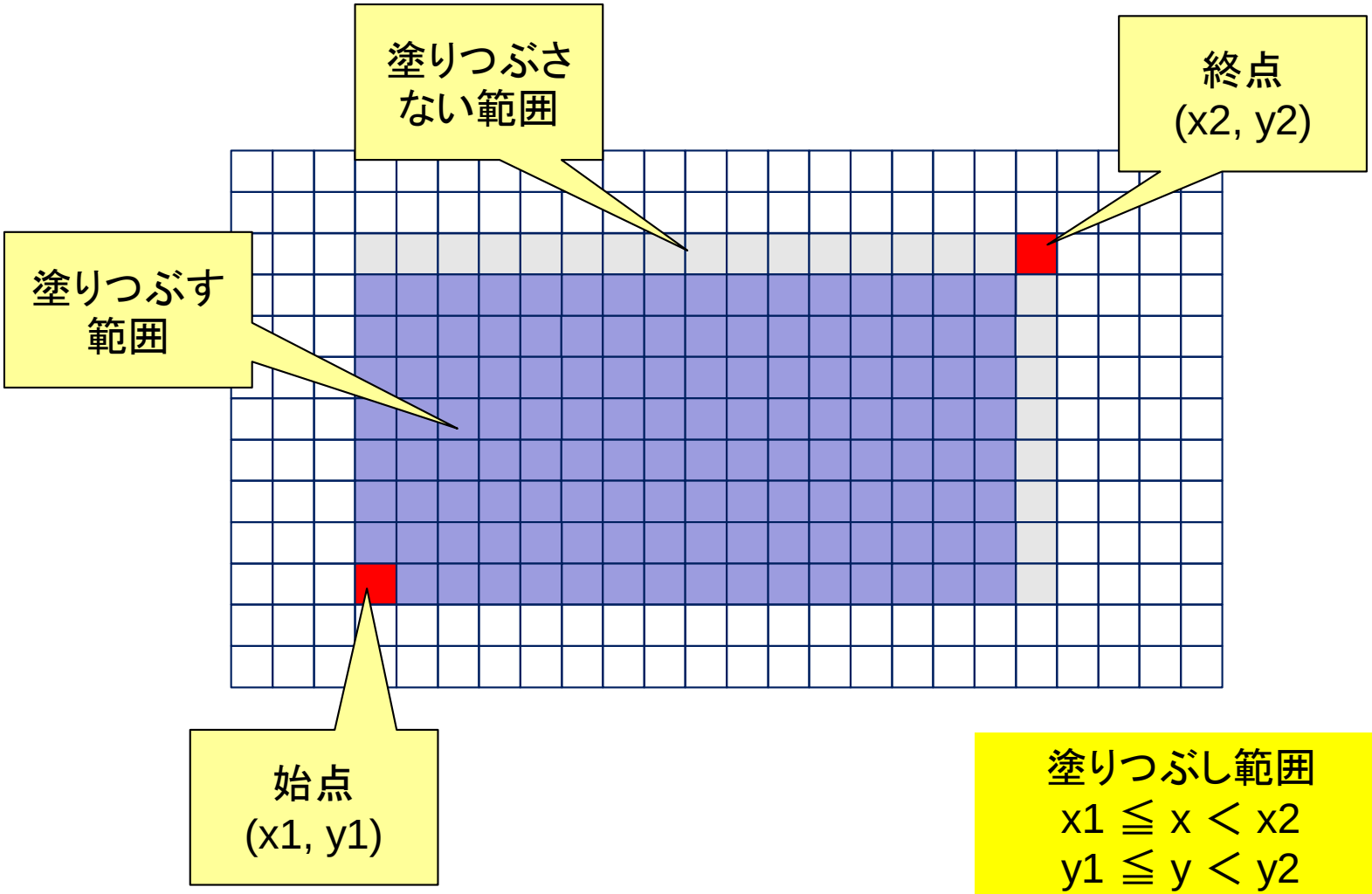
・(1)と(2)の両方をWordに貼り付けて提出！

・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



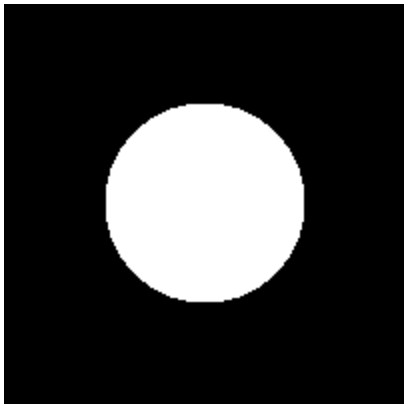
発展課題3

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

・(1)スクリーンショットと(2)BMPファイルの両方をWordに貼り付けて提出！

・実行例を二つ以上提出！

発展課題3のヒント

(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

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

で表される．