

プログラミング演習II

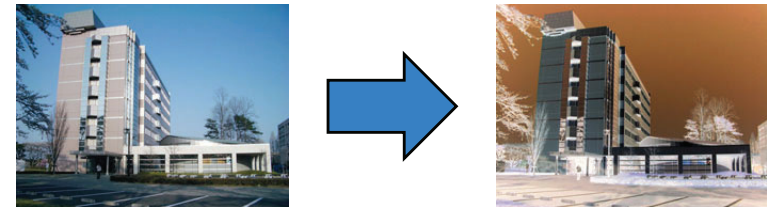
課題4 画像処理

第2週目

担当: 篠田一馬

課題4の概要

- 画像処理プログラムを作成することで, ファイル入出力, コマンドライン引数, ポインタ配列, ビット演算を学ぶ



2

全体の流れ

- 1週目 (ファイル操作, コマンドライン引数)
 - 画像ファイルのヘッダ情報をテキストファイルに出力
- 2週目 (ファイル操作)
 - 画像ファイルの画素情報をバイナリファイルに出力
- 3週目 (ポインタ配列と動的確保)
 - 様々な画素数の画像ファイルの入出力に対応
- 4週目 (ビット演算)
 - 画像を減色または上下反転させて出力

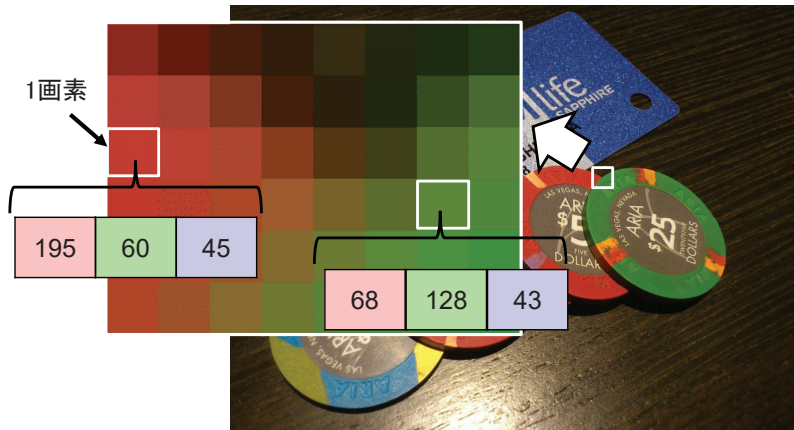
3

テキストファイルとバイナリファイル

4

デジタル画像データの基本

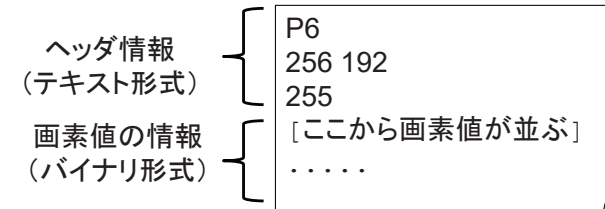
- 画素ごとに赤, 緑, 青の3色の情報を持っている
- 赤, 緑, 青(RGB)の組み合わせで色を表現



5

PPM画像(P6)

- 画像に関する情報をヘッダとして持つ画像
 - 縦画素数と横画素数の情報が含まれる
 - 余計な情報が少ないため, 処理しやすい

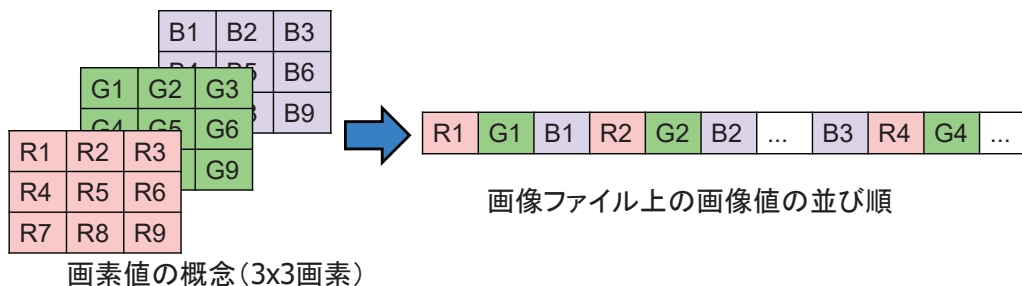


ファイル上のデータ

6

画素値の情報

- 基本的にR, G, Bの順に数値が一行に並ぶ
 - メモリは1次元なので, 1次元に展開される
 - 画素値はテキストではなく**バイナリデータ**として保存



画像ファイル上の画像値の並び順

画素値の概念(3x3画素)

7

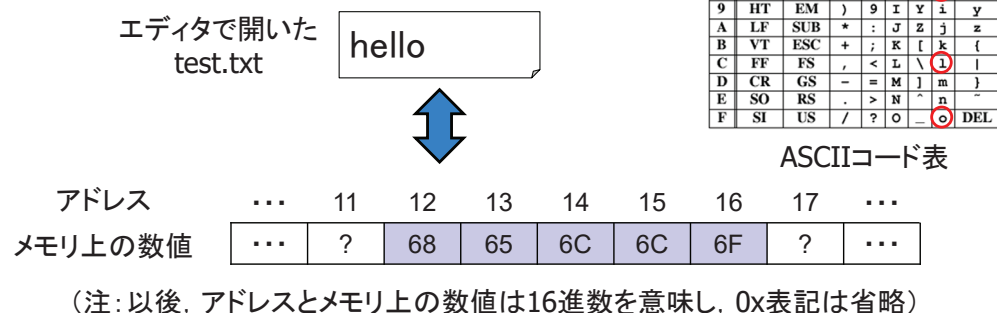
テキストファイルとバイナリファイル

- テキストファイル
 - 一般的には, ASCIIコード表で文字に変換できる数値(0x00から0x7F, つまり符号なし整数だと0から127)のみが含まれるファイル
- バイナリファイル
 - それ以外のファイル
 - PPMファイル(P6)はバイナリファイル

8

テキストファイル

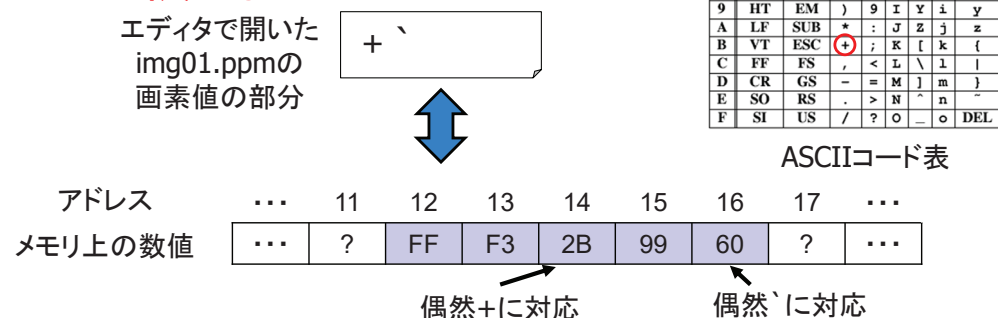
- メモリ上はバイナリデータ(数値)が保存される
- テキストエディタで開くと, ASCIIコードに対応している文字を表示



9

バイナリファイル

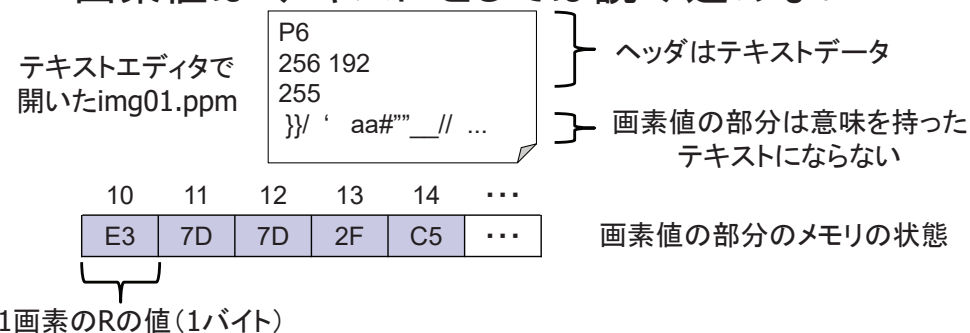
- メモリ上はバイナリデータ(数値)が保存される
- テキストエディタで開いたときに, ASCIIコードの文字に対応するとは限らない



10

PPMファイルはバイナリファイル

- 1画素の各色を1バイトで表現
- 画素値は0から255(0x00から0xFF)なので, ASCIIコード表の文字とは対応しない
- 画素値は"テキスト"としては読み込めない



11

バイナリデータの
読み込みと書き込みの例

12

バイナリデータの読み書き

■ テキストデータであれば下記が簡単

- fscanf関数: scanfと同じ書式で文字を読み込み
- fgets関数: 改行までテキスト1行文を読み込み
- fprintf関数: printfと同じ書式で文字を書き込み

■ バイナリデータの場合

- fread関数: 指定されたバイト数のデータをファイルから読み込み
- fwrite関数: 指定されたバイト数のデータをファイルに書き込み

13

fread関数の使用例

ファイル用のメモリ領域

10	11	12	13	14	...
E3	7D	7D	2F	C5	...

}}/' aa#"__// ...

input.dat
(バイナリファイル)

```
int main(void)
{
    FILE *fp;
    char *fname = "input.dat";
    char buffer[100];

    fp = fopen(fname, "rb");

    fread(buffer, sizeof(char), 100, fp);

    fclose(fp);

    return 0;
}
```

14

fread関数の使用例

ファイル用のメモリ領域

10	11	12	13	14	...
E3	7D	7D	2F	C5	...

}}/' aa#"__// ...

input.dat
(バイナリファイル)

```
int main(void)
{
    FILE *fp;
    char *fname = "input.dat";
    char buffer[100];

    fp = fopen(fname, "rb");

    fread(buffer, sizeof(char), 100, fp);

    fclose(fp);

    return 0;
}
```

} バイナリファイルの読み込みなので
バイナリモード"rb"を指定

15

fread関数の使用例

ファイル用のメモリ領域

10	11	12	13	14	...
E3	7D	7D	2F	C5	...

}}/' aa#"__// ...

input.dat
(バイナリファイル)

```
int main(void)
{
    FILE *fp;
    char *fname = "input.dat";
    char buffer[100];

    fp = fopen(fname, "rb");

    fread(buffer, sizeof(char), 100, fp);

    fclose(fp);

    return 0;
}
```

30 31 32 33 34 ...
E3 7D 7D 2F C5 ...

main関数のbufferの
メモリ領域

} このfread関数の呼び出しで、
ファイルからbufferに1バイトの
要素を100個読み込む

16

fread関数の使用例

ファイル用のメモリ領域

10	11	12	13	14	...
E3	7D	7D	2F	C5	...

input.dat
(バイナリファイル)

main関数のbufferの
メモリ領域

30	31	32	33	34	...
E3	7D	7D	2F	C5	...

【fread関数】
第一引数: 格納先のポインタ
第二引数: 1要素あたりのバイト数
第三引数: 要素数
第四引数: ファイルポインタ

```
int main(void)
{
    FILE *fp;
    char *fname = "input.dat";
    char buffer[100];

    fp = fopen(fname, "rb");

    fread(buffer, sizeof(char), 100, fp);

    fclose(fp);

    return 0;
}
```

17

fread関数の使用例

ファイル用のメモリ領域

10	11	12	13	14	...
E3	7D	7D	2F	C5	...

input.dat
(バイナリファイル)

main関数のbufferの
メモリ領域

30	31	32	33	34	...
E3	7D	7D	2F	C5	...

【sizeof関数】
引数に型を指定すると、その型の
バイト数を返す。一般的には、
char, unsigned char: 1
int: 4

```
int main(void)
{
    FILE *fp;
    char *fname = "input.dat";
    char buffer[100];

    fp = fopen(fname, "rb");

    fread(buffer, sizeof(char), 100, fp);

    fclose(fp);

    return 0;
}
```

18

fread関数の使用例

```
int main(void)
{
    FILE *fp;
    char *fname = "input.dat";
    char buffer[100];

    fp = fopen(fname, "rb");

    fread(buffer, sizeof(char), 100, fp);

    fclose(fp);

    return 0;
}
```

main関数のbufferの
メモリ領域

30	31	32	33	34	...
E3	7D	7D	2F	C5	...

あとは、bufferの各要素の数値(画素値)
を使って、出力ファイルへのコピーや
数値の編集(画像処理)を行えばよい

19

fwrite関数の使用例

```
int main(void)
{
    FILE *fp, *fpo;
    char *fname = "output.dat";
    char buffer[100];
    /*bufferにデータが入力されたと想定*/

    fpo = fopen("output.dat", "wb");

    fwrite(buffer, sizeof(char), 100, fpo);

    fclose(fpo);

    return 0;
}
```

main関数のbufferの
メモリ領域

30	31	32	33	34	...
E3	7D	7D	2F	C5	...

20

fwrite関数の使用例

```
int main(void)
{
    FILE *fp, *fpo;
    char *fname = "output.dat";
    char buffer[100];
    /*bufferにデータが入力されたと想定*/

    fpo = fopen("output.dat", "wb");

    fwrite(buffer, sizeof(char), 100, fpo);

    fclose(fpo);

    return 0;
}
```

30	31	32	33	34	...
E3	7D	7D	2F	C5	...

main関数のbufferの
メモリ領域

読み込み時とほぼ同じ。モードは"wb",
fread関数ではなくfwrite関数を使用
(引数ほぼ同じ, bufferからfpoに出力)

21

fwrite関数の使用例

```
int main(void)
{
    FILE *fp, *fpo;
    char *fname = "output.dat";
    char buffer[100];
    /*bufferにデータが入力されたと想定*/

    fpo = fopen("output.dat", "wb");

    fwrite(buffer, sizeof(char), 100, fpo);

    fclose(fpo);

    return 0;
}
```

20	21	22	23	24	...
E3	7D	7D	2F	C5	...

ファイル用のメモリ領域

}}/ aa#"__// ...

output.dat
(バイナリファイル)

30	31	32	33	34	...
E3	7D	7D	2F	C5	...

main関数のbufferの
メモリ領域

読み込み時とほぼ同じ。モードは"wb",
fread関数ではなくfwrite関数を使用
(引数ほぼ同じ, bufferからfpoに出力)

22

(補足)構造体のsizeof

■ 構造体の確保に必要な総バイト数を返す

- **sizeof(PIXEL)**: (一般的には)3
- PIXEL構造体のデータを
freadやfwriteで読み書きしたい
場合は便利

```
typedef struct {
    unsigned char r;
    unsigned char g;
    unsigned char b;
} PIXEL;
```

■ 余分な領域を確保(パディング)して偶数長や4の倍数になる処理系もある

- 右のPIXEL構造体は合計7バイトに思えるが, 多くの処理系では
sizeof(PIXEL)は8を返す

```
typedef struct {
    unsigned char r;
    unsigned char g;
    unsigned char b;
    int a;
} PIXEL;
```

23

(補足)char型の変数について

■ charは**整数**が格納される変数

- int型と違うのは, サイズが1バイト(-127から128)
- unsigned char型も1バイトで, 0から255

```
int main(void)
{
    char a, b, c;

    a = 3;      /* OK */
    b = -2;     /* OK */
    c = a + b; /* OK, cは1 */

    return 0;
}
```

24

(補足)なぜ文字型というのか

- ASCIIコードの数値は1バイトで、文字コードを格納するのが便利だから

- 'h'は右表の通り0x68に対応
- プログラム中の'h'は自動的に0x68(十進数で104)に変換される
- aには104が代入される

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	@	P	`	p	
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	=	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

ASCIIコード表

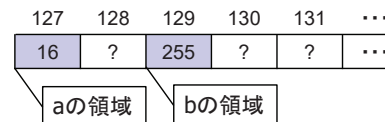
```
int main(void)
{
    char a;
    a = 'h';    /* OK */
    return 0;
}
```

(補足)10進数と16進数

- 左右はどちらも全く同じ意味であることに注意
 - 資料やプログラムを書くときにわかりやすい方を使用

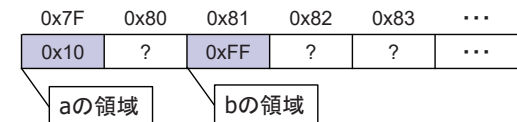
10進数表記

```
int main(void)
{
    unsigned char a, b;
    a = 16;
    b = 255;
    return 0;
}
```



16進数表記

```
int main(void)
{
    unsigned char a, b;
    a = 0x10;
    b = 0xFF;
    return 0;
}
```



画素値の読み込みと書き込み

プログラム上での画素値の読み書き

- 画素値を入れるための領域は静的に確保済

```
typedef struct {
    unsigned char r;
    unsigned char g;
    unsigned char b;
} PIXEL;

typedef struct {
    int    xsize;
    int    ysize;
    int    level;
    PIXEL pBuffer[49152];
} IMAGE;
```

画素値を格納するための構造体

img01.ppm専用に、49152要素
(256 × 192画素)のPIXELの配列を用意

プログラム上での画素値の読み書き

■ 画素値の読み込みは未実装

```
int iioLoadFile(IMAGE* pImage, char* fname)
{
    FILE* fpr;
    char lineBuffer[LINEMAX];
    int i;

    /* ファイルオープン（ひな形で実装済） */
    if ((fpr = fopen(fname, "rb")) == NULL) {
        .....
        sscanf(lineBuffer, "%d", &pImage->level);
        /* ここでヘッダ部読み込み終了 */

        /* pImage->pBufferに画像データを格納（2週目） */

        /* ファイルクローズ（ひな形で実装済） */
        fclose(fpr);
        return 0;
    }
}
```

} ファイルからpImage->pBufferに画素値を読み込むだけ。

29

プログラム上での画素値の読み書き

■ 画素値のコピーは未実装

```
void ipCopy(IMAGE* p_in_image, IMAGE* p_out_image)
{
    /* ヘッダ情報のコピー */
    p_out_image->xsize = p_in_image->xsize;
    p_out_image->ysize = p_in_image->ysize;
    p_out_image->level = p_in_image->level;

    /* ここで画素値をコピーする */
}
```

} ヘッダ情報はコピーしているが、画素値の情報（pBufferの各要素）はコピーしていない。

30

プログラム上での画素値の読み書き

■ 画素値の書き込みは未実装

```
int iioSaveFile(IMAGE* pImage, char* fname)
{
    int i;
    FILE* fpo;

    /* ファイルオープン（1週目で実装済） */

    /* PPMのヘッダ情報をファイルに出力（1週目で実装済） */
    /* 画素値をファイルに出力 */

    /* ファイルクローズ（1週目で実装済） */

    return 0;
}
```

} ヘッダ出力後、pImage->pBufferをファイルに書き込むだけ。

31

【演習課題4_2_1】

- ファイルからの画素値の読み込み、およびファイルへの書き込みを実装
- 前週の課題からの変更点
 - iioLoadFileとiioSaveFileの実装
 - iioSaveFileのファイルopenはwbモードにすること
 - ipCopyにおける画素値のコピーの実装
- 入力画像と同じ画像が出力できるか確認する
 - 実行時の出力ファイル名の拡張子をtxtからppmにすること

32

演習課題の目安: 授業終了時刻まで

終わった人は

- 余裕があれば次週の課題に取り組むことを推奨
- 最終レポートは、画像処理の実装種類数が多いほど評価UP
 - 例年, 10種類以上実装する人も数名
- 作業が終わったら, 次週に備えてUSBメモリ等にプログラムをコピーすること