

バイナリデータの 読み込みと書き込みの例

1

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

2

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;
}
```

3

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);

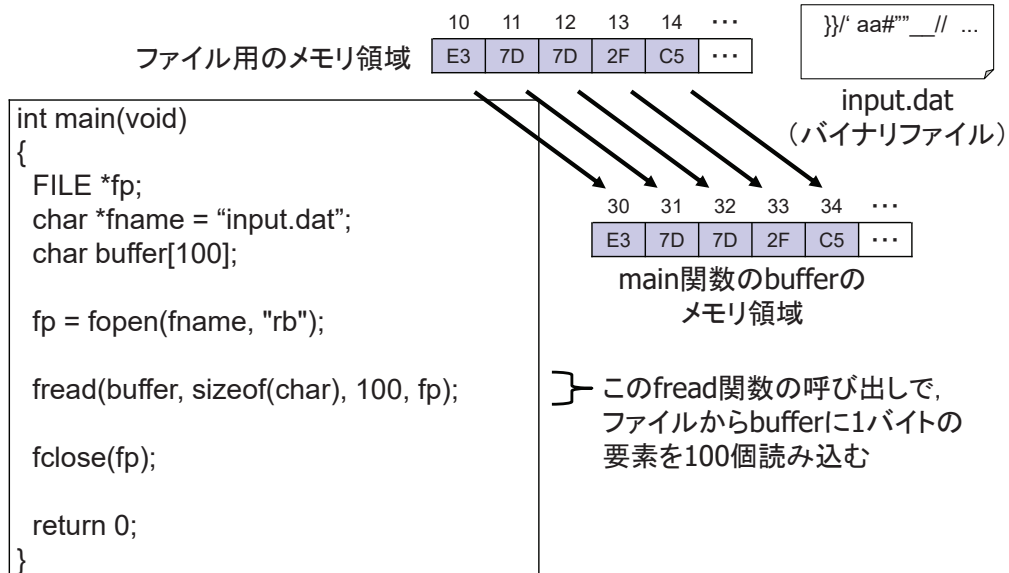
    fclose(fp);

    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

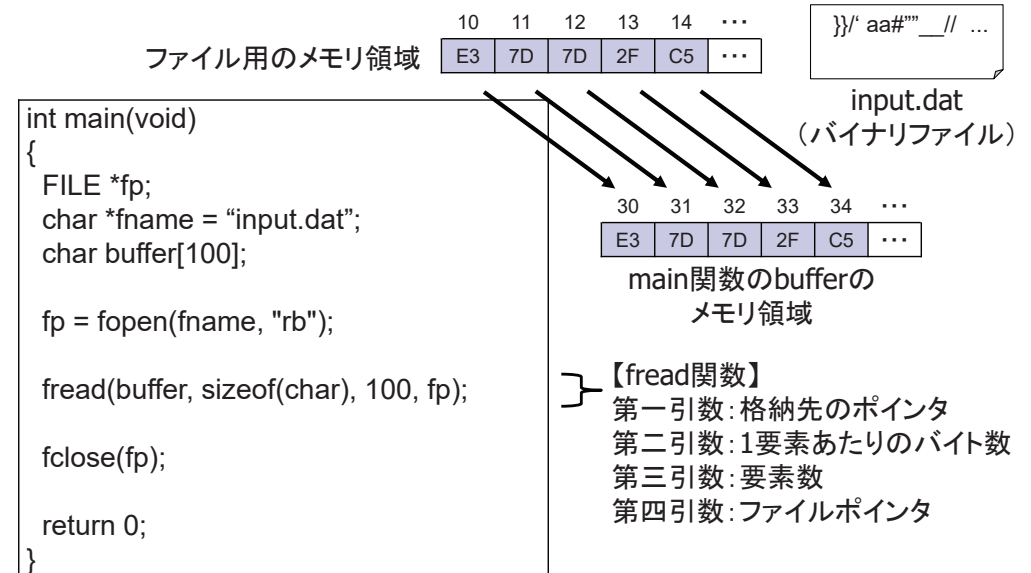
4

fread関数の使用例



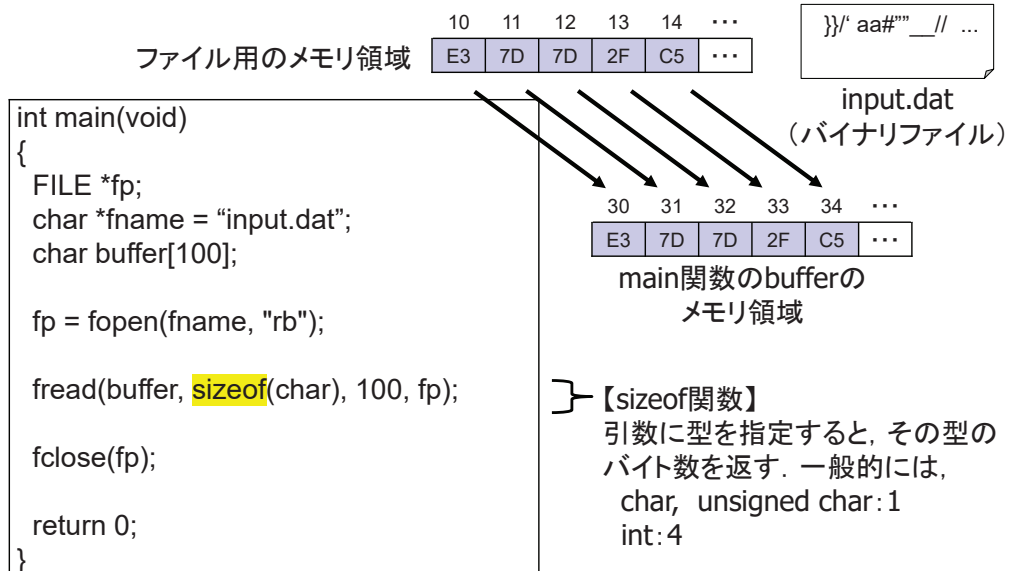
5

fread関数の使用例



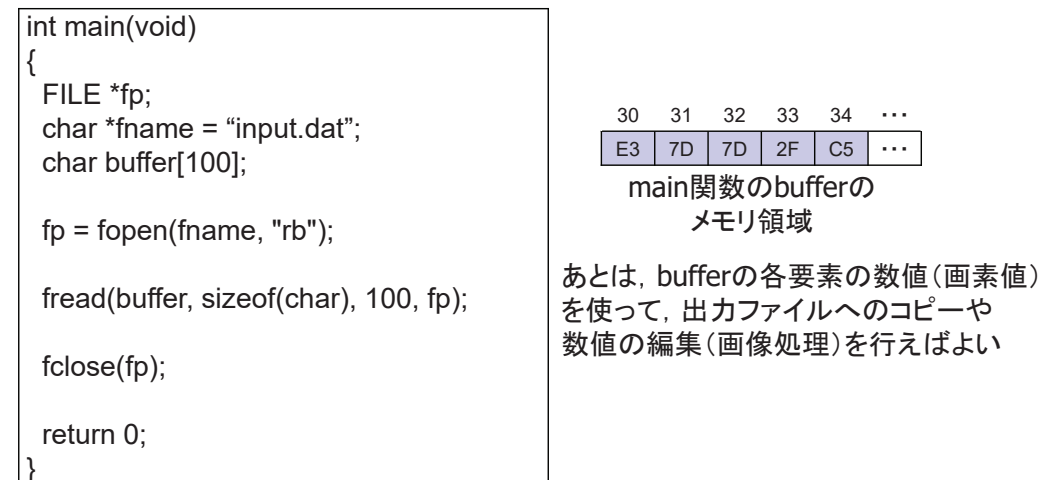
6

fread関数の使用例



7

fread関数の使用例



8

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

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)

fwrite関数の使用例

ファイル用のメモリ領域

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

output.dat
(バイナリファイル)

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

```
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

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)

(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;
```

(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;
}
```

13

(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;
}
```

14

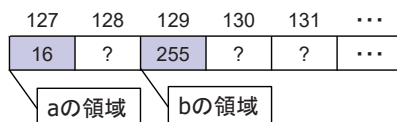
(補足)10進数と16進数

■ 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

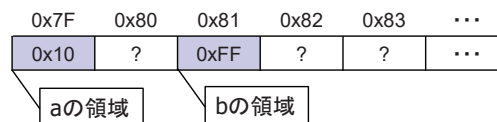
10進数表記

```
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;
}
```



15