プログラミング発展

2023年度2Q　火曜日5~7時限(13:45~16:30)
金曜日5~7時限(13:45~16:30)

工学院　情報通信系

尾形わかは，松本隆太郎，
Chu Van Thiem, Saetia Supat
TA:東海林郷志，千脇彰悟

第9回「ファイル入出力の基礎」

1. 前回のおさらいと課題の解説
2. ファイルへの入出力：
   バイナリファイルの扱い

ファイル入出力の基礎

バイナリファイルの扱い

ファイルの入出力

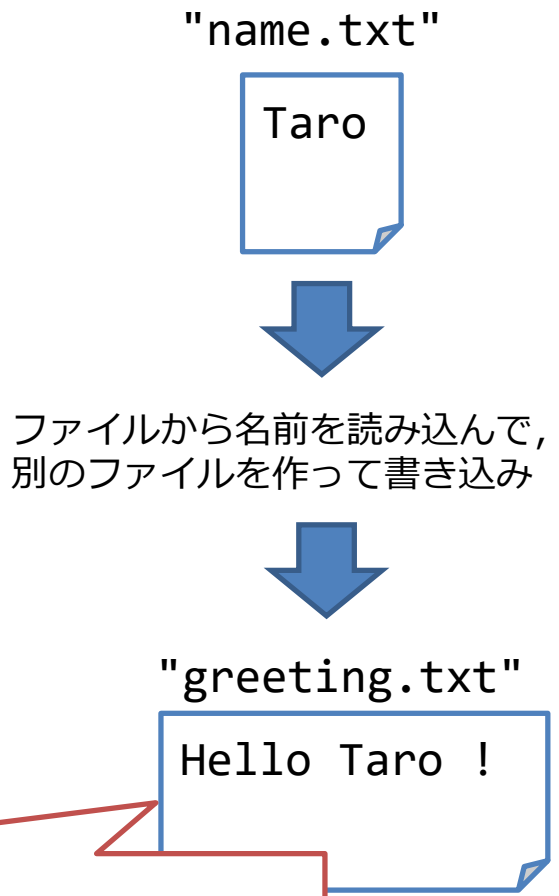
ファイルの使い方例

```
int main()
{
    FILE *fp_in, *fp_out;
    char name[10];

    fp_in = fopen("name.txt", "r");
    fscanf(fp_in, "%s", name);
    fclose(fp_in);

    fp_out = fopen("greeting.txt", "w");
    fprintf(fp_out, "Hello %s !¥n", name);
    fclose(fp_out);

    return 0;
}
```



ファイル＝どこかに保存されてる電子データ．
実際には，ファイルには何がどのように保存されているの？

ファイル

ファイル　＝　所詮は 0と1の並び．その０と１の並びが，何を表しているのか？は，その時々によって異なる．

00000000001111000010111011111111 ……

　　8ビット（＝1バイト）ごとに区切って処理されるのが普通．

00000000 00111100 00101110 11111111....

　　見ずらいので，8ビット（＝1バイト）を2桁の16進数で表す

00 3c 2e ff ....

3cが，数値60を表すのか，文字'<' を表すのか，あるいは

　　003C2eff で一つの数値を表すのか．．．

それは，ファイルを作った人と，それを使う人との取り決めによる．

テキストファイル

「文字が並んでいる」と解釈するファイル．Macの「テキストエディット」，windowsの「メモ帳」などで表示・編集できる．
文字⇔0,1列 の対応もいろいろあるが，代表的なものはUTF-8.

- **UTF-8**：世界で使われている文字や絵文字などを，1～複数バイトで表現．
  最も基本的な文字（次ページ，ASCII文字と呼んだりする）は1バイトで表現．
  日本語の漢字は大抵３バイト

```
fprintf(fp, "%s", "Hello, world");
```

によって，テキストファイルに

HのASCIIコード

```
48 65 6c 6c 6f 2c 77 20 6f 72 6c 64
```

（16進表記）

が書き込まれる．ASCII文字の1文字＝1バイト．

```
fprintf(fp, "%d",10);
```

であれば，　31 30　が書き込まれる．（2文字なので２バイト）

```
fprintf(fp, "%x",16);
```

10進数の 16 は，16進数では 10 なので．

の場合も　31 30　が書き込まれる．　（2文字なので２バイト）

ASCII コード表

‘A’のコードは，　16進数で　40 (縦軸）＋1（横軸）＝41

★制御文字 ★サロゲートペア ★合成文字 ★未定義

	US-ASCII - us-ascii															
	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0																
10																
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
128文字																

制御文字

普通の文字

https://uic.jp/charset/show/us-ascii/

テキストファイル

「文字が並んでいる」と解釈するファイル．Macの「テキストエディット」，windowsの「メモ帳」などで表示・編集できる．
文字⇔0,1列 の対応もいろいろあるが，代表的なものはUTF-8．

- **UTF-8**：世界で使われている文字や絵文字などを，1～複数バイトで表現．
  最も基本的な文字（次ページ，ASCII文字と呼んだりする）は1バイトで表現．
  日本語の漢字は大抵３バイト

```
fprintf(fp, "%s", "Hello, world");
```

によって，テキストファイルに

48 65 6c 6c 6f 2c 77 20 6f 72 6c 64　　（16進表記）

が書き込まれる．ASCII文字の1文字＝1バイト．

HのASCIIコード

1バイト（８ビット）は，0～255の整数を表現できるはずなのに，10ですら２バイトになってしまう

```
fprintf(fp, "%d",10);
```

であれば，　31 30　が書き込まれる．（2文字なので２バイト）

```
fprintf(fp, "%x",16);
```

10進数の 16 は，16進数では 10 なので．

の場合も　31 30　が書き込まれる．（2文字なので２バイト）

バイナリファイル

- バイナリファイル：文字に限らず，任意のデータを含むファイル.

- int型であれば，1つの値をsizeof(int)バイトで保存できる.
  - sizeof(int)=4　なので，-2,147,483,648 ～ 2,147,483,647 の範囲の整数値を，４バイトでファイルに保存できる.
  - int型の数を1000個保存したファイルは，4000バイトになる.

- char型であれば，　1つの値を1バイトで保存
  - -128～127 の範囲の整数値を，1バイトでファイルに保存
  - ASCII文字１文字＝１バイト

ファイルを扱う関数： fopen, fclose

```
FILE *fopen(   // 正常のときファイルポインタ，エラーのときNULL
    char *filename,     // 開きたいファイルの名前（パス名）
    char *mode          // モード（→次ページ参照）
);
```

例：
```
    FILE *fp1=fopen("sample.txt","r+");
        //カレントディレクトリにあるファイルを開く
    FILE *fp2=fopen("folder/sample.dat","rb");
        //カレントディレクトリのfolderの中にあるファイルを開く
        //絶対パスでも相対パスでもOK
```

ファイルの「パス名」やカレントディレクトリが分からない人は，
1回目講義の参考資料を参考のこと．わからなければ質問してください．

```
int fclose(         //  正常のとき０，エラーのときEOF
    FILE *fp        // ファイルポインタ
);
```

開いたファイルは閉じよう．

モード

モード	動作	ファイルがあるとき	ファイルがないとき
"r"	読み込み専用	正常	エラー（NULL返却）
"w"	書き込み専用	サイズを 0 にする（上書き）	新規作成
"r+"	読み込みと書き込み	正常	エラー（NULL返却）
"w+"	書き込みと読み込み	サイズを 0 にする（上書き）	新規作成

- バイナリの時は，rb, wb, ab, rb+, wb+ などとする．
- 他にも，追加書き込み用の “a” と “a+” があるが，使う場面が少ないので以降では考えない．

バイナリファイルを扱う関数： fread

ファイルから読み込む：

```
size_t fread(  //  戻り値は，読み込んだ変数の個数
    void *ptr,          // 読み込んだ値（要素）を格納するポインタ
    size_t size,        // 要素１個あたりのバイト数
    size_t n,           // 読み込む要素の個数
    FILE *fp            // ファイルポインタ
);
```

正常に読み込めれば，nと一致.
ファイルが終了するなどで読み込めない
場合はnより小さい値が返る

配列に読み込む場合以外は１

例

```
int x, count;
char str[10];
FILE *fp = fopen(....);
count = fread(&x, sizeof(int), 1, fp);
count = fread(str, sizeof(char), 10, fp);
```

厳密には型キャストが必要.
`count =(int) fread(...)`

長さ10の配列の場合，一気に10個読み込める

バイナリファイルを扱う関数： fwrite

ファイルに書き出す：

```
size_t fwrite(      //  戻り値は，書き込めた変数の個数
    void *ptr,          // 書き込む値（要素）を格納しているポインタ
    size_t size,        // 要素１個あたりのバイト数
    size_t n,           // 書き込む要素の個数
    FILE *fp            // ファイルポインタ
);
```

正常に書き込めればnと一致
エラーなどで書き込めない場合はnより小さい値が返る

配列を書き込む場合以外は１

例

```
    int x, count;
    char str[10];
    FILE *fp = fopen(....);
    count = fwrite(&x, sizeof(int), 1, fp);
    count = fwrite(str, sizeof(char), 10, fp);
```

長さ10の配列の場合，一気に10個書き込める

ファイル位置表示子

- 原則，freadはファイルの頭から順番に読み込んでいく．
- 例：ファイルの内容が

```
  41 11 ff ff 11 42 43 00
```

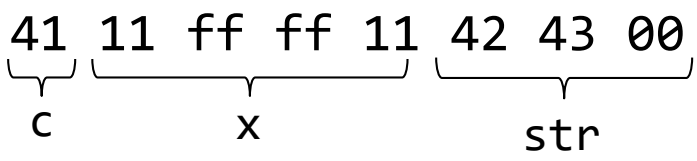
であるとき，このファイルから

```
  fread(&c, sizeof(char), 1, fp);
  fread(&x, sizeof(int), 1, fp);
  fread(str, sizeof(char), 3, fp);
```

でデータを読むと，

```
 c = 'A'
 x = 0x11ffff11
 str = "BC"
```

となる．



- fwriteで書き込むときも同様．連続して書き込んでいく．

- ファイル位置表示子：現在，読んで・書いているところを示す値．fopenした直後のファイル位置表示子は，通常，ファイルの先頭を示している．fread，fwrite，などによって値が増える．

バイナリファイルを扱う関数： fseek

ファイル位置表示子を好きなところへ移動させる関数：

```
int fseek(          // 上手くいけば戻り値は0
    FILE *fp,
    long offset,          // 基準位置からの移動バイト数
    int origin            // 基準位置
);
```

・基準位置：次の３つの定数が定義されている.

SEEK_SET	ファイルの始め
SEEK_CUR	現在のファイル位置表示子の値
SEEK_END	ファイルの終わり（※）

※バイナリファイルにおいては SEEK_ENDは正しく
ファイルの終わりを示さない場合がある

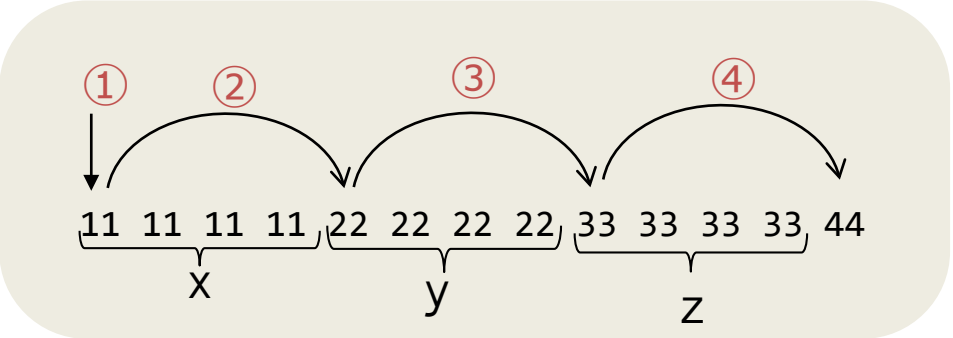
fseekの使用例：

```
int main(){
    FILE *fp;
    int x,y,z;

    if ((fp = fopen("sample.data","rb+"))==NULL) return 1;    ①
    fread(&x, sizeof(int), 1, fp);   ②
    fread(&y, sizeof(int), 1, fp);   ③
    fread(&z, sizeof(int), 1, fp);    ④

    x=4*x; y=4*y; z=4*z;
    fseek(fp, 0, SEEK_SET);    // ファイルの先頭に戻る.
    fwrite(x, sizeof(int), 1, fp);
    fseek(fp, sizeof(int), SEEK_CUR);    //  int 一つ分とばす
    fwrite(z, sizeof(int), 1, fp);

    fclose(fp);
    return 0;
}
```



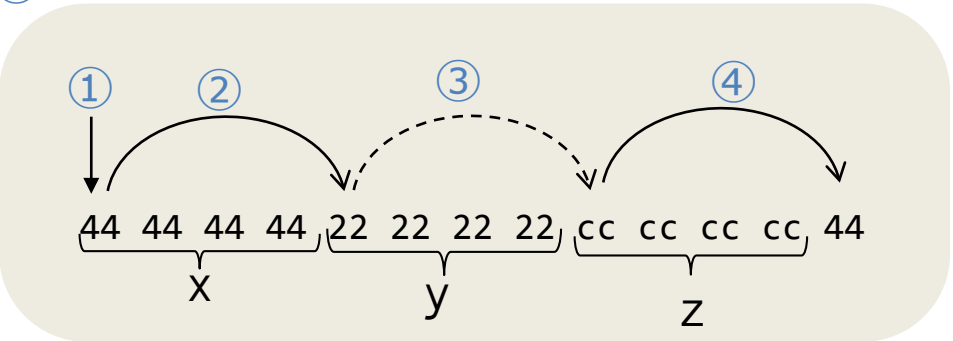
fseekの使用例：

```
int main(){
    FILE *fp;
    int x,y,z;

    if ((fp = fopen("sample.data","rb+"))==NULL) return 1;    ①
    fread(&x, sizeof(int), 1, fp);   ②
    fread(&y, sizeof(int), 1, fp);   ③
    fread(&z, sizeof(int), 1, fp);    ④

    x=4*x; y=4*y; z=4*z;
    fseek(fp, 0, SEEK_SET);    // ファイルの先頭に戻る.  ①
    fwrite(x, sizeof(int), 1, fp);     ②
    fseek(fp, sizeof(int), SEEK_CUR);    //  int 一つ分とばす   ③
    fwrite(z, sizeof(int), 1, fp);   ④

    fclose(fp);
    return 0;
}
```



Ex9-0

シーザー暗号の暗号化をするプログラムを作りなさい.

- **シーザー暗号**： ‘A’ → ‘D’, ‘B’ → ‘E’, ..., ‘X’ → ‘A’, ‘Y’ → ‘B’, ‘Z’ → ’C’ のように，各アルファベットを「3文字ずらす」ように変換する．元の文がN文字ならば，暗号文もN文字.
- **前提**：ファイル “ex9_0_plain.txt” には，A～Z，カンマ，スペース，ピリオドなどの文字が並んでいるとする.

プログラムの仕様：

- “ex9_0_plain.txt” から１行ずつ文字列を読み込み，各文字に対して
  - ‘A’, ‘B’, ..., ‘Z’ ：シーザー暗号で暗号化
  - それ以外：変更しない

  とし，新しいファイル “ex9_0_cipher.txt” に書き出す.
- “ex9_0_plain.txt” に書かれている文字列によらず動作するようにすること． １行の文字数は，1000文字未満と仮定してよい.

★提出物：プログラム：ex9_0.c

Ex9-0

- ファイルから１行ずつ文字列を読みこむ：
  ex4-2で使った，以下を参考にするとよい.

```
char instr[BUF_SIZE];
while (fgets(instr, BUF_SIZE, infp) != NULL)
```

- １行分のファイルへの書き出しは

```
fgets(instr, outfp);
```

でできる.

Ex9-1

テキストファイルの読み書きをしてみよう

① テキストファイルに，1,000個のランダムな整数を書き込むプログラムを作りなさい．仕様は以下の通り．

1．ファイルを モード "w" で開く．
　出力するファイルの名前は，ex9_1.txt とすること．
　（ファイル名が違うと採点ができない可能性がある．）
2．乱数を srand(1); で初期化する．
3．以下を1,000回繰り返す：
　０以上 10,000未満の整数xをランダムに生成し， xを10進数としてファイルにfprintfで書きこむ．
4．ファイルを閉じる．

② ①で作成したテキストファイルをモード "r" で開き，1,000個の整数を fscanfで読み込み，和を計算して表示するプログラムを作りなさい．

★提出物：書き込みのプログラム：ex9_1_1.c
　　読み込みのプログラム：ex9_1_2.c

Ex9-1　（つづき）

注意：

- ①　作成したファイル(ex9_1.txt）のサイズを予想し，予想どおりであるかを確認してみること.

- ②　出力（整数の和）がどの程度の値になるのかを予想し，出力がおよそ予想通りとなることを確認してみること.

fprintf

printf をファイルなどにも書き出せるように一般化した関数.

```
int fprintf(   //戻り値は，出力したバイト数（ASCII文字ならば文字数）
    FILE *fp,       //ファイルポインタ
    char *format,  //出力フォーマット．Printfと同じ
    ...)           //出力する値の列．Printfと同じ
```

例）

```
FILE *fp=fopen("data.txt","w");
fprintf(fp,"Hello world¥n");
```

とすれば，data.txtというファイルに書き出す.

ファイル以外にも書き出せる.

fprintf(stdout,…);　は　printf(…); と同じ.

※stdout (standard output)＝標準出力＝画面とか

fscanf

scanf をファイルなどからも読み込めるように一般化した関数.

（以前も指摘したように，fscanfにはセキュリティ上の問題があるが，簡単のため，今日の課題では使ってください）

```
int fscanf(   //戻り値は，読み込んだ変数の数
    FILE *fp,       //ファイルポインタ
    char *format,  //出力フォーマット．scanfと同じ
    ...)           //出力する値の列．scanfと同じ
```

例）

```
FILE *fp=fopen("data.txt","r");
fscanf(fp,"%s %d", str, &x);
```

とすれば，data.txtというファイルから，文字列strと十進数xを読み込む.

ファイル以外からも読み込める.

fscanf(stdin,…);　は　scanf(…); と同じ.

※stdin (standard input)＝標準入力＝キーボードとか

Ex9-2

・バイナリファイルを扱ってみよう.

① バイナリファイルに，1,000個のランダムな整数を書き込むプログラムを作りなさい.

1. ファイルをモード "wb" で開く.
   出力するファイルの名前は，ex9_2.dat とすること.
   （ファイル名が違うと採点ができない可能性がある.）
2. 乱数を srand(1); で初期化する.
3. 以下を1,000回繰り返す：
   ０以上10,000未満の整数をランダムに生成し，short int型の変数 x に代入し，
   fwrite(&x, sizeof x, 1, fp);
   によってファイルに書きこむ.
4. ファイルを閉じる.

② ①で作成したファイルをモード "rb" で開き， 1,000個の整数を読み込み，和を計算して表示するプログラムを作りなさい.

★提出物：書き込みのプログラム：ex9_2_1.c
読み込みのプログラム：ex9_2_2.c

Ex9-2 (つづき)

注意：

- Ex9-1のコードを少し改変すればできるはず・・・
- 乱数の生成方法は，Ex9-1と揃えること.

- ①　作成されるバイナリファイル(ex9_2.dat)のサイズを予想し，予想どおりであるかを確認してみること.

- ②　出力（整数の和）が，Ex9-1と等しいことを確認すること.（乱数のシードが同じであるから，同じ1,000個の乱数が生成されるはずである．）

Ex9-3

・ファイルの中身を表示するツールを自作してみよう.

①ファイル名を引数として受け取り，ファイルの内容を表示する関数：

```
int file_dump(char *filename)
```

を実装しなさい．この関数の仕様は以下のとおり.

a. ファイルをモード"rb"でfopenし，ファイルが存在しない場合（fopenがNULLを返した場合）は，"No file" と表示した後，0以外の戻り値(値は適宜)で終了.

b. fopenが成功した場合は，ファイルから16バイトずつ読み込み，以下の２通りで表示する.

i. 各バイトを，16進２桁で表示.

ii. 各バイトを，ASCII文字1文字で表示．ただし，印字できない文字（ASCIIコード表の水色部分以外）の場合は空白文字を表示.

次ページの出力例のように，16バイトを１行で表示し，２通りの表示を左右に並べる.

ファイルを最後まで表示するか，20行分（320バイト）を表示したら，ファイルをfcloseし，0を戻り値として終了.

②ファイルの内容を file_dump 関数を用いて表示するプログラムを作りなさい.

- 標準入力からファイル名を読み込む
- 取得したファイル名を file_dump 関数に与えて実行

Ex9-3 (つづき)

各バイトを1文字で表示，
印字できない場合はスペースとする．
16文字分

各バイトを16進２桁で，16バイト分

出力例：

```
filename? : test.txt
30 31 32 33 34 35 36 37 38 39 0A 41 42 43 44 45  | 0123456789 ABCDE |
46 47 20 61 62 63 64 65 66 67                    | FG abcdefg |
```

0x20 はスペース．印字している

0x0A は改行を表す制御文字．
スペースを表示

windowsの場合，0A（改行）のところが 0D0A かもしれない．

- 提出物：
  - ソースファイル　（ex9_3.c）

Ex9-3 （補足説明）

- １バイトのデータを扱いたい場合，unsigned char型が便利.
- freadの戻り値を見れば，16バイト読み込んだのか，その前にファイルが終了したのかが分かる.
- 印字できる文字なのかを判定する標準関数isprintがある.
- MacOSの xxdコマンド が，本課題の出力の左側と同様の結果を出すので，確認に使える.

発展課題

- ファイルサイズ（バイト数）が16の倍数で無い場合は，最後の行には適当にスペースを入れて見やすくする. （出力例のように）
- 20行（320バイト）表示して終了するのではなく，いったん表示を停止して more ? と表示して，'y'以外が入力されたらそこで終了. 'y'が入力されたら，さらに20行を表示する.
（大きいファイルも，途中で止めながら先を見られるように. ）

今日の提出物まとめ

- Ex9-0
  - ex9_0.c
- Ex9-1
  - ① ex9_1_1.c ← テキストで書き出し
  - ② ex9_1_2.c ← テキストで読み込み
- Ex9-2
  - ① ex9_2_1.c ← バイナリで書き出し
  - ② ex9_2_2.c ← バイナリで読み込み
- Ex9-3：
  - ex9_3.c （発展課題の機能を追加すれば加点）

注意！

- mallocで確保したメモリを freeするのと同様に，
  fopenしたファイルは，fcloseすること！
- fopenする際には，
  　テキストファイル → テキストモード
  　バイナリファイル → バイナリモード
  を選ぶのが普通であるが，
  テキストファイルをバイナリモードで fopen したり，
  バイナリファイルをバイナリモードで fopen することも可能
- モードで何が違うか？　テキストモードでは，ファイルの読み書きの際に，改行コード（OSによって異なる）等の変換が自動的に行われる．バイナリモードでは，そのような変換は無い
- Ex9-3では，どんなファイルであってもファイル内容を表示できるようにしたいので，バイナリモードを使ってください

コピペレポートについて

プログラムや考察などが他の提出者と重複している場合、不正とみなして減点および問い合わせをすることがあります