プログラミング第一同演習

慶應義塾大学 理工学部 情報工学科
講義担当: 河野 健二
演習担当: 杉浦 裕太

試験について

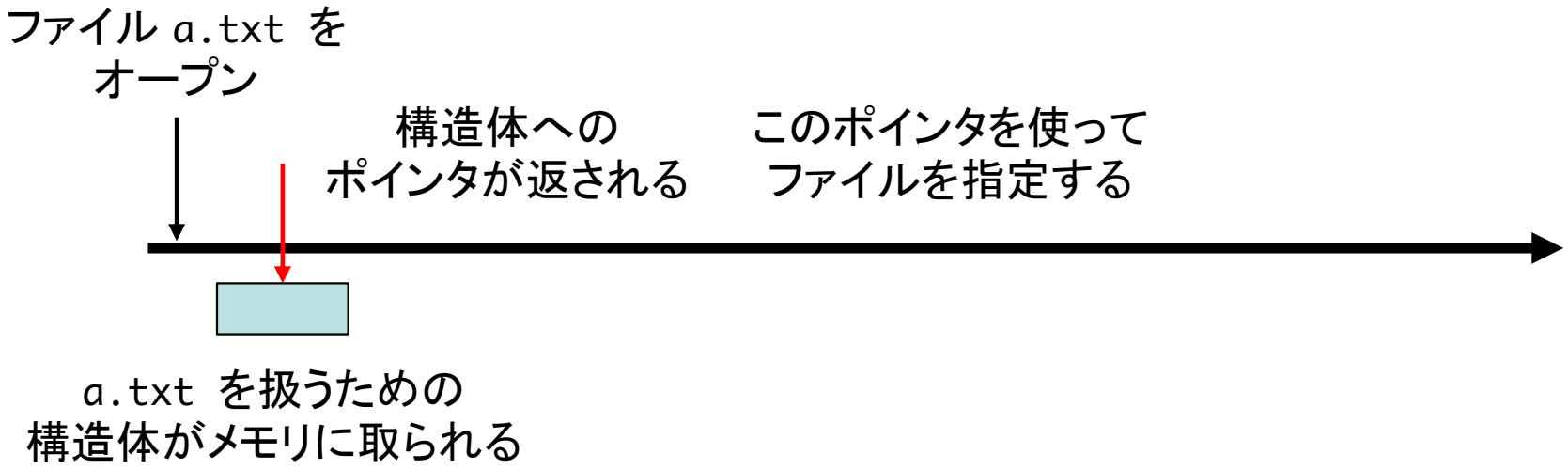
- 日時・場所
  - 学生課の掲示にしたがって下さい
  - 時間を間違えないこと
- 試験内容：
  - 筆記試験（コンピュータは使わない）
  - C 言語の基本について問う
    - 基本的な文法が理解できているか
    - 配列，構造体，ポインタが理解できているか
    - 簡単な C のプログラムが書けるか
- 持ち込み不可

本日の内容

- ファイルの取り扱い
  - fopen, fclose
  - fgets, fputs
  - fgetc, fputc
  - fprintf
  - fseek
  - Windows 環境での注意
- 少しすすんだ話
  - バッファリング
  - fflush
  - 標準入出力，標準エラー出力

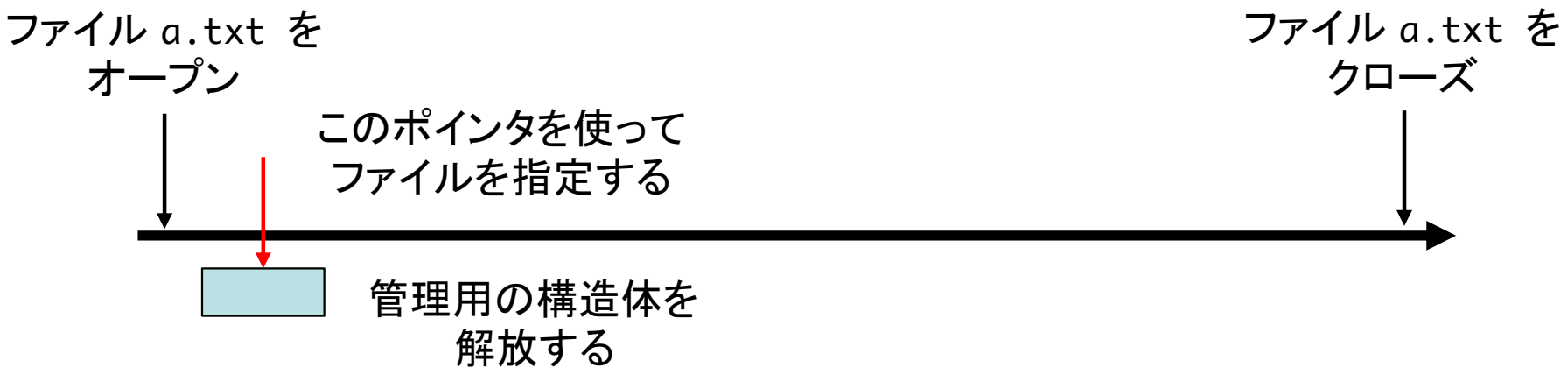
ファイルの扱い（1/2）

- ファイルは オープン (*open*) してから読み書きする
  - ファイル名（正確にはパス名）を指定してオープンする
    - 指定したファイルを管理する構造体が割り当てられる
  - その結果，その構造体へのポインタが返される
    - 読み書きの際に，そのポインタを使って対象のファイルを指定する



ファイルの扱い（2）

- 使い終わったら クローズ (*close*) する
  - 割り当てられたデータ構造を解放する



FILE 構造体

- ファイルをオープンした時に割り当てられる構造体
  - FILE 構造体の中身は気にしなくてよい
- ファイル a.txt をオープンすると・・・
  - a.txt を扱うための FILE 構造体がメモリに割り当てられ，
  - その FILE 構造体を指すポインタが返ってくる
- 別のファイル b.txt をオープンすると・・・
  - b.txt を扱うための・・・（以下同様）

fopen: ファイルをオープンする


```
FILE *fopen(char *path, char *mode)
```

- path: オープンするファイルのパス名
- mode: ファイルを扱うモード
  - 読み出し専用（書き込み不可）なのか，読み書き可能なのか，など
  - 詳細は後で解説する

- 戻り値:
  - FILE 構造体へのポインタ
    - FILE 構造体がメモリのどこかに割り当てられる
    - 割り当てられた構造体へのポインタが返る
  - ファイルがオープンできないときは NULL を返す
    - 読み込みたいファイルが存在しないときなど

fopen: モードについて (1/2)

- FILE *fopen(char *path, char *mode)
  - path: オープンするファイルのパス名
  - mode: ファイルを扱うモード

- ファイルに対して行う操作を指定する
  - "r" : 読み出し専用でオープン．書き込みは不可
    - ファイルの先頭から順に読み込む
    - オープン時にファイルがなければエラー（NULL が返る）
  - "w" : 書き込み専用でオープン．読み出しは不可
    - ファイルの先頭から書き込む
    - オープン時にファイルがなければ，空のファイルが作られる

fopen: モードについて (2/2)

sslab system software

- "a" ：書き込み専用にオープン．読み出しは不可
  - ファイルの終端から書き込む（追記をする）
  - オープン時にファイルがなければ，空のファイルが作られる
- "r+" ：読み書き用にオープン
  - ファイルの先頭から順に読み書きを行う
  - オープン時にファイルがなければ，エラー
- "w+"：読み書き用にオープン
  - ファイルの先頭から書き込む
  - オープン時にファイルがなければ，空のファイルが作られる
- "a+"：読み書き用にオープン
  - ファイルの終端から書き込む
  - オープン時にファイルがなければ，空のファイルが作られる

fopen: 簡単な利用例

- ファイル "sample.txt" を読出し専用でオープン

```
#include <stdio.h>

int main() {
        FILE *fp;
        …
        fp = fopen("sample.txt", "r");
        if (fp == NULL) {
                // エラーメッセージを表示
                exit(1);
        }
        …

}
```

```
if ((fp = fopen("sample.txt", "r")) == NULL) {
     // エラーメッセージを表示
     exit(1);
}
```
と書いてもよいことに注意

fclose: ファイルをクローズする

- int fclose(FILE *fp)
  - fp: クローズする FILE 構造体へのポインタ
  - 戻り値
    - クローズに成功すれば 0
    - 失敗すると EOF という特別な値
- オープンしたファイルが不要になったら呼び出すこと
  - メモリ上に確保された FILE 構造体を解放する
  - なお，同時にオープンできるファイル数には限りがある
    - 理論上は，クローズしないと新しくファイルをオープンできなくなる

fclose: 簡単な利用例

- オープンしたファイルをクローズ

```
#include <stdio.h>

int main() {
        FILE *fp;
        …
        if ((fp = fopen("sample.txt", "r")) == NULL) {
                // エラーメッセージを表示
                exit(1);
        }
        …
        fclose(fp);
        …
}
```

fgetc: 1 バイトの読み込み

```
int fgetc(FILE *fp)
```

- オープンされたファイル fp から 1 バイトを読み込む

- 戻り値
  - 1 バイトの値を 0 ～ 255 の範囲で返す
    - 1 バイトの値を（符号なしで）int 型に拡張する
  - ファイルの終端に達したとき，またはエラーが発生したとき：
    - EOF という特別な値を返す

プログラム例（1）：ファイルの内容を表示

- （英文の）テキストファイルの内容を表示する
  - バイナリファイルや日本語のファイルに適用すると，おかしなことになるので注意

```
#include <stdio.h>
int main() {
        FILE *fp;
        int c;
        if ((fp = fopen("sample.txt", "r")) == NULL) {
                // エラーメッセージを表示
                exit(1);
        }
        while ((c = fgetc(fp)) != EOF)
                printf("%c", c);
        fclose(fp);
        return 0;
}
```

変数 c にファイルの内容を 1 バイトずつ読み込む
ファイルの終端に達すると EOF が返る

プログラム例（2）：ファイルの内容を表示

- バイナリファイルの内容を 16 進数で表示する
  - printf で変数 c の値の表示の仕方を変えるだけ
  - フォーマット指定 "%02x" を用いて 16 進数 2 桁で表示

```
#include <stdio.h>
int main() {
        FILE *fp;
        int c;
        if ((fp = fopen("sample.txt", "r")) == NULL) {
                // エラーメッセージを表示
                exit(1);
        }
        while ((c = fgetc(fp)) != EOF)
                printf("%02x", c);
        fclose(fp);
        return 0;
}
```

ここを変えるだけでよい

fgetc と似たもの

- fgetc と似た関数（あるいはマクロ）がある
  - マクロについては補足のスライドを参照のこと
  - 今日の段階では関数と同じと思っても差し支えない

- int getc(FILE *fp)
  - fgetc と本質的には同じ
    - getc はマクロとして定義されている点だけが違う

- int getchar()
  - 常に標準入力から読み込む
    - getc(stdin) と同じ．stdin については講義の最後で説明する

perror: エラー原因の表示

- void perror(char *msg)
  - エラーが発生したとき，「msg + ": " + エラー原因」を表示する

例：

```
#include <stdio.h>
int main() {
        FILE *fp;
        if ((fp = fopen("sample.txt", "r")) == NULL) {
             // エラーメッセージを表示
             perror("sample.txt");
             exit(1);
        }
        …
```

実行例：指定されたファイル（sample.txt）が存在しない時

```
% ./a.out
sample.txt: No such file or directory
```

fputc: 1 バイトの書き込み

- `int fputc(int c, FILE *fp)`
  - オープンされたファイル `fp` に 1 バイトの値を書き込む
  - 引数の `c` は書き込む 1 バイトの値を指定する
    - `c` の値は 0 以上 255 以下の整数を指定する
    - それ以外の値を指定すると，不思議なことが起きることがある
  - 戻り値
    - 書き込んだ値を返す
    - ただし，エラーが発生したとは `EOF` を返す

プログラム例：大文字・小文字変換

- 入力ファイルを a.txt，出力ファイルを b.txt とする

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

char *a = "a.txt";
char *b = "b.txt";

int main()
{
    int c;
    FILE *fp_a, *fp_b;
```

```
    if ((fp_a = fopen(a, "r")) == NULL) {
        perror(a);
        exit(1);
    }

     if ((fp_b = fopen(b, "w")) == NULL) {
            perror(b);
            exit(1);
     }

     // 次のスライドの続く
```

プログラム例：大文字・小文字変換

- int toupper(int c)
  - 小文字を大文字に変換
  - 引数 c が小文字なら，
    大文字の ASCII コードを返す
- int tolower(int c)
  - 大文字を小文字に変換
  - 引数 c が大文字なら，
    小文字の ASCII コードを返す

```
    while ((c = fgetc(fp_a)) != EOF) {
        if ('a' <= c && c <= 'z')
            fputc(toupper(c), fp_b);
        else if ('A' <= c && c <= 'Z')
            fputc(tolower(c), fp_b);
        else
            fputc(c, fp_b);
      }

      fclose(fp_a);
      fclose(fp_b);

      return 0;
}
```

fgets: 行単位での読み込み

sslab system software

`char *fgets(char *str, int size, FILE *fp)`

オープンされたファイル `fp` から行単位で読み込み

- 引数:
  - `str` で指定した配列に 1 行単位で読み込みを行う
    - 改行文字 (`'¥n'`) を読み込んだところでやめる
    - 改行文字も読み込まれる
  - ただし，読み込むサイズは最大 (`size` − 1) バイトまで
    - 読み込んだ内容の末尾にヌル文字 (`'¥0'`) を追加するため
- 戻り値:
  - 読み込みに成功したとき: 読み込んだ文字列へのポインタを返す
  - ファイルの終端に達したとき，またはエラーが発生したとき: NULL を返す

fgets: 使い方

sslab system software

- 十分な大きさの char 型の配列を用意する
  - `char buffer[80];`
- fgets を用いて buffer に読み込みを行う
  - `// fp はオープンされたファイル`
    `// 配列 buffer の要素数は 80 なので，最大 80 バイト読み込む`
    `fgets(buffer, 80, fp);`
- 配列の大きさに注意！
  - `char buffer[32];`
    `fgets(buffer, 33, fp);` `// 間違い！`
    `// メモリの内容を壊し，タチの悪いバグとなる`

fgets: 動作例

- 次のファイルを読み込んだとしよう
  - わかりやすくするために改行コードを明示的に示してある

```
a¥n
ab¥n
abc¥n
de
```

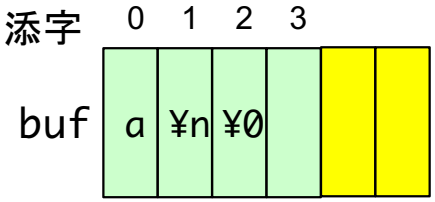
  - 次のように fgets を用いて順次，読み込んでいく

```
char buf[4];      // 説明のため小さめ
fp = fopen(…);
fgets(buf, 4, fp);
fgets(buf, 4, fp);
fgets(buf, 4, fp);
fgets(buf, 4, fp);
fgets(buf, 4, fp);
fgets(buf, 4, fp);
```

fgets を実行するたびに buf の内容がどうなるか考えてみよう

fgets: 動作例（1/3）

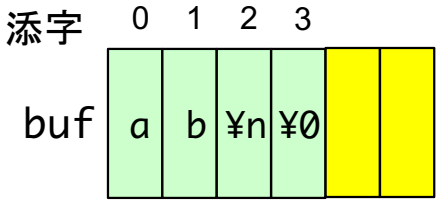
- 1 回目の fgets を実行
  - 改行までを読み込み，ヌル文字を追加



a¥n
ab¥n
abc¥n
de

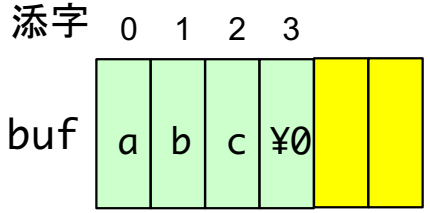
読み込むファイル

- 2 回目の fgets を実行
  - 改行までを読み込み，ヌル文字を追加



fgets: 動作例 (2/3)

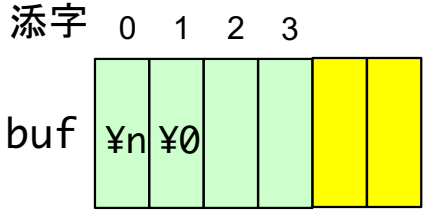
- 3 回目の fgets を実行
  - 読み込める最大 3 文字を読み，ヌル文字を追加



追加されるヌル文字を入れて，最大 4 文字

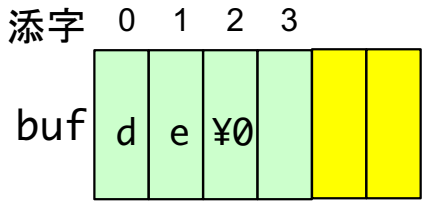
a¥n
ab¥n
abc¥n
de

- 4 回目の fgets を実行
  - 残りの改行を読み込み，ヌル文字を追加



fgets: 動作例（3/3）

- 5 回目の fgets を実行
  - ファイルの終端まで読み，ヌル文字を追加



a¥n
ab¥n
abc¥n
de

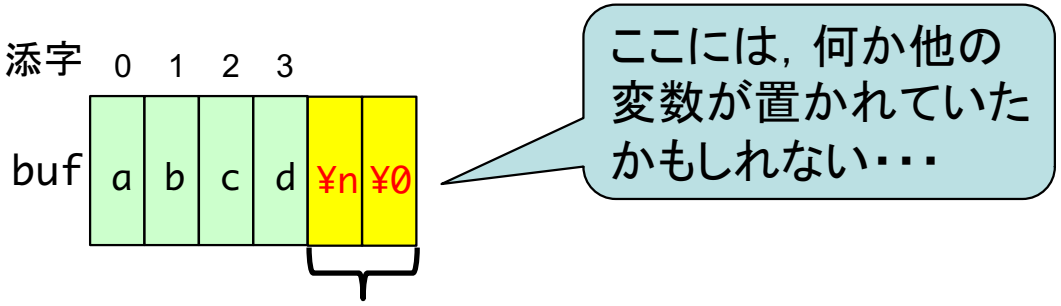
- 6 回目の fgets を実行
  - ファイルの終端まで読んだため，NULL を返す

fgets: やってはいけないこと

- 次のようなプログラムを書くと・・・
  - char buf[4];
    …
    fgets(buf, 6, fp);
    …

abcd¥n
abcd¥n

読み込むファイル



配列の範囲を超えて書き込まれてしまう

fputs: 文字列の書き出し

```
int fputs(char *str, FILE *fp)
```

- 引数：str で指定した文字列をファイル fp に書き出す
- 戻り値:
  - 書き出しに成功したとき: 非負の整数を返す
  - 書き出しに失敗したとき: EOF を返す

fgets & fputs: プログラム例

- ファイルの各行を読み込み，表示し，別のファイルにコピーする
  - ただし，各行は改行文字を入れても 79 文字以下とする

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

char *a = "a.txt";
char *b = "b.txt";
char buffer[80];

int main()
{
    int c;
    FILE *fp_a, *fp_b;

    if ((fp_a = fopen(a, "r")) == NULL) {
        perror(a);
        exit(1);
    }
        if ((fp_b = fopen(b, "w")) == NULL) {
            perror(b);
            exit(1);
    }

    while (fgets(buffer, 80, fp_a) != NULL) {
            printf("%s", buffer);
            fputs(buffer, fp_b);
    }
    fclose(fp_a);
    fclose(fp_b);
    return 0;
}
```

fprintf: printf のファイル対応版

```
int fprintf(FILE * fp, char *format, …)
```

int printf(char *format, …) の最初に引数 fp を追加した感じ

- 画面に表示する代わりに，引数 fp でしたファイルに出力する
  - 例：

```
int x = 10, y = 100;
…
fp = fopen("sample.txt", "w");
…
fprintf(fp, "x = %d, y = %d¥n", x, y);
```

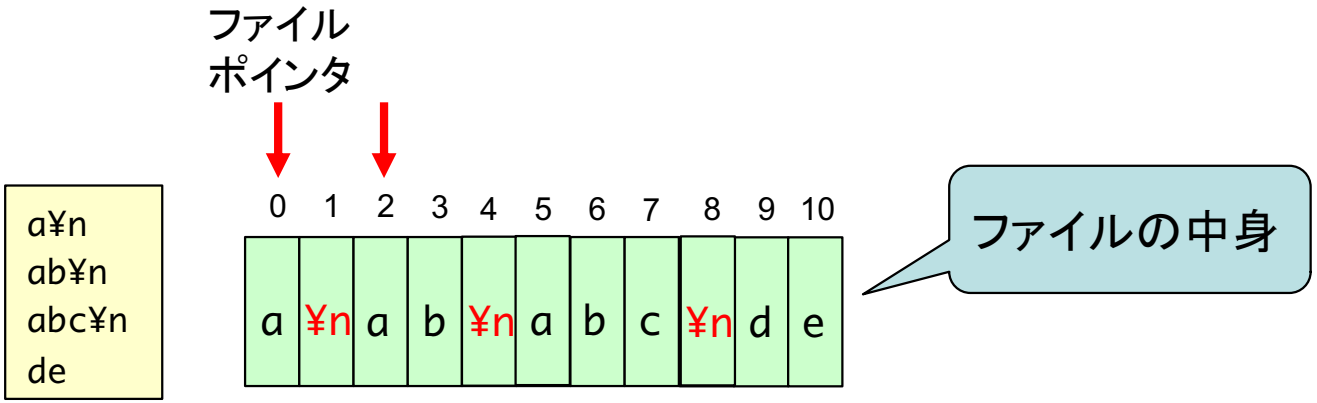
sample.txt への出力結果

```
x = 10, y = 100¥n
```

画面に出力する代わりに，
ファイルに出力する

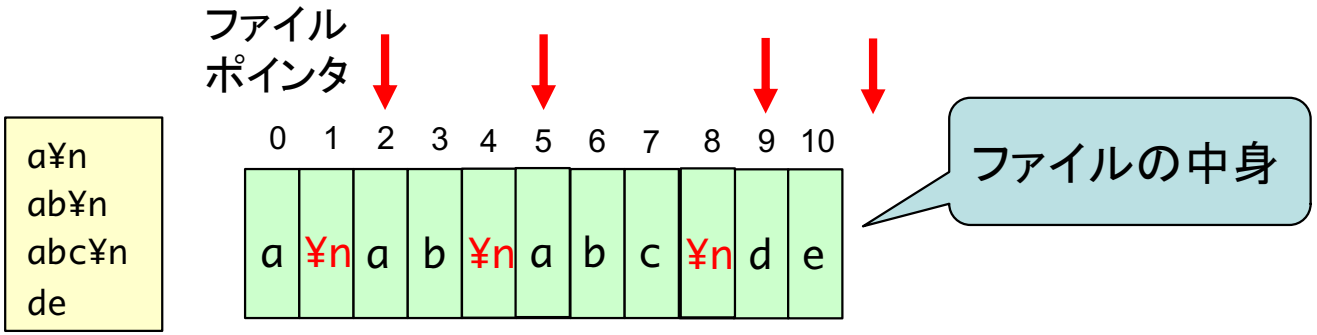
ファイルポインタ (1/2)

- 次に読み書きするファイル上の場所をさす
  - C 言語の “ポインタ” とはまったくの別物
- 例:
  - ファイルを "r" モードでオープンする
    - ファイルポインタはファイルの先頭を指す
  - fgets を実行する
    - 改行まで読み込まれる. ファイルポインタは改行の次まで進む



ファイルポインタ (2/2)

- さらに fgets を実行する
  - 改行まで読み込まれる．ファイルポインタは改行の次まで進む
- さらに fgets を実行する
  - 改行まで読み込まれる．ファイルポインタは改行の次まで進む
- さらに fgets を実行する
  - ファイルの終端まで読み込む．ファイルポインタは終端の先を指す
- さらに fgets を実行する
  - ファイルポインタはファイルの終端の先にあるので，NULL を返す



fseek: ファイルポインタの操作

sslab system software

```
int fseek(FILE *fp, long offset, int whence)
```

ファイルポインタの位置を変更する

- 引数
  - fp: 操作対象のオープンされたファイル
  - offset: 何バイト（マイナスも可）ずらすかを指定する
  - whence: どこから offset バイトだけずらすのかを指定する
    - SEEK_SET: ファイルの先頭から
    - SEEK_CUR: 現在のファイルポインタの位置から
    - SEEK_END: ファイルの終端から
- 戻り値:
  - 成功なら 0, 失敗なら -1

ftell: ファイルポインタの位置の取得

long ftell(FILE *fp)

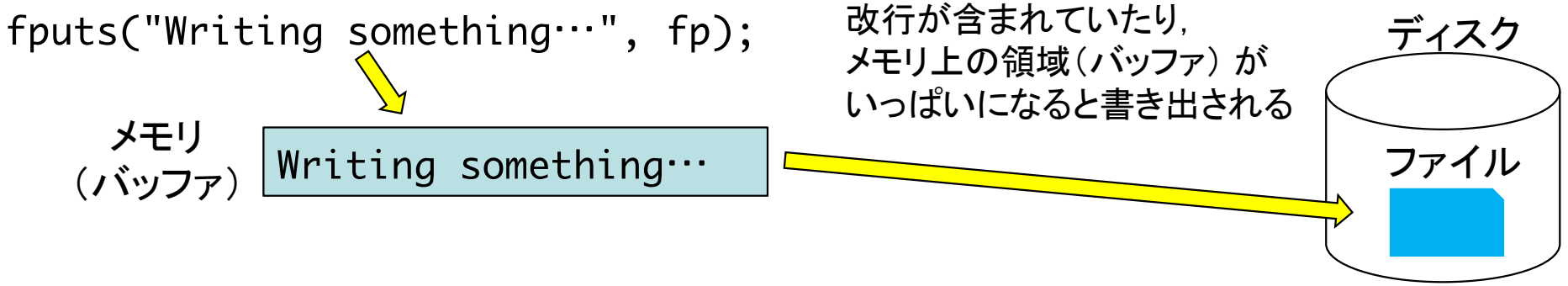
ファイル fp の現在のファイルポインタの位置を返す

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

int main()
{
    FILE *fp;
    long sz;
    if ((fp = fopen("sample.txt", "r")) == NULL) { perror("sample.txt");  exit(1);  }

    fseek(fp, 0, SEEK_END);  // ファイルポインタを終端に移動
    sz = ftell(fp);          // ファイルポインタの位置を取得
    printf("File size = %d¥n", sz);

    return 0;
}
```

ファイルの大きさ（バイト数）が求まる

進んだ話題

- バッファリング（buffering）
  - ファイルへの出力を（メモリ上に）ためておくこと
    - 例：fputs しても，すぐにファイルには出力せず，メモリにためておく
  - つまり，ファイルへの出力はすぐに行われるとは限らない
    - ファイル入出力は遅いため，なるべくまとめて行おうとする
    - 3 年生の「オペレーティングシステム」あたりで学ぶ



実験：バッファリングの様子

- 画面出力も（ファイルと同様）バッファリングされる

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    printf("Hello¥n");      // 改行があるため，すぐに出力される
    printf("World");        // 改行がないため，すぐには出力されない
    sleep(3);               // 3 秒ほどねる （実行を止める）
    printf("!!!!¥n");       // 改行があるため，ここでまとめて出力される
    return 0;
}
```

- 実行結果

```
Hello
World!!!!
```

3 秒たってから，!!!!¥n と一緒に表示される

バッファリングのせいで，World はすぐに表示されない

fflush: 強制出力

int fflush(FILE *fp)

バッファリングされている内容を出力する

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    printf("Hello¥n");
    printf("World");
    fflush(stdout);          // 画面への出力を強制する．stdout は標準出力
    sleep(3);                // 3 秒ほどねる （実行を止める）
    printf("!!!!¥n");
    return 0;
}
```

- 実行結果

```
Hello
World!!!!
```

fflush による強制出力

3 秒たってから出力

標準出力と標準入力

- あらかじめオープンされている FILE 構造体がある
- 標準入力
  - キーボードから入力に対応する
  - stdin という変数があらかじめ用意されている
    - stdin の型は FILE *
- 標準出力
  - 画面への出力に対応する
  - stdout という変数があらかじめ用意されている
    - stdout の型は FILE *

- ファイルから読み込むのと同様に，stdin から読み込める
- ファイルに出力するのと同様に，stdout に出力できる

標準エラー出力

- あらかじめオープンされているもう一つのFILE構造体
- 標準エラー出力
  - 画面への出力に対応する
  - stderr という変数があらかじめ用意されている
    - stderr の型は FILE *
  - 画面に出力するのと同様に，stderr に出力できる
    - エラーメッセージの表示に使うのが普通
    - シェルの使い方を学ぶと，stdout/stderr を分ける利点がわかる
- 標準エラー出力と標準出力 (stdout) との違い
  - stdout はバッファリングされる
  - stderr はバッファリングされない（すぐに画面に出力される）

標準入出力・エラー出力の使い方

sslab system software

- オープンされたファイルと同じように使える
  - `fgets(buffer, 80, stdin)`
    - キーボードからの読み込み
  - `fputs(buffer, stdout)`
    - 画面への出力
  - `fprintf(stdout, "Hello")`
    - `printf("Hello")` と同じ
  - `fprintf(stderr, "Error:…")`
    - 標準エラー出力にメッセージを表示
    - 画面に表示されるという点では `stdout` と似ている

feof と ferror (1/2)

- fgetc の仕様を思い出そう

```
int fgetc(FILE *fp)
```

- 戻り値
  - 正しく読めたき：読み込んだ 1 バイトの値を 0 ～ 255 の範囲で返す
  - ファイルの終端に達したとき，またはエラーが発生したとき：EOF を返す

- fgetc の戻り値が EOF だったら？
  - ファイルの終端またはエラー
  - どうやって区別するの？
    - ファイルの終端なら，最後までファイルが読めたので問題ない
    - エラーだったら，エラーメッセージを表示したい

feof と ferror (2/2)

- int feof(FILE *fp)
  - ファイルの終端に到達していたら
    - ゼロ以外の値を返す
  - ファイルの終端ではなかったら
    - ゼロを返す
- int ferror(FILE *fp)
  - エラーが発生していたら
    - ゼロ以外の値を返す
  - エラーが発生していなかったら
    - ゼロを返す

```
while ((c = fgetc(fp)) != EOF)
            printf("%c", c);

// この時点ではエラーが起きたのか,
// ファイルの終端に達したのか不明

if (ferror(fp)) {
        // エラーが発生していたら，エラー処理
        perror("…");
        exit(1);
}
```

Window 環境での注意

- ITC でのみ作業をする人には無関係！

- Windows では Linux とは違う改行コードを使っている
- そのため，ファイル読み書きの際に改行コードを自動で変換する

- 改行コードを変換されると困るときは？
  - バイナリモードを使う
  - モードに b を追加する
    - "rb", "wb", "ab" といった具合

- Linux や（最近の）Mac OS では気にしなくてよい

他にもたくさんある！

sslab system software

- 入出力を扱う標準関数は他にたくさんある
  - sscanf() など，紹介していないが有用

- 自分でどんどん調べて覚えること
  - 最初は Google 等で検索した情報でもよい
  - ただし，書いている人が誤解していることも多い
  - 一番正確なのは，man コマンドを使うこと
    - man fopen
      などとやると，fopen のマニュアルが表示される
    - 内容が正確なため，初心者にはとっつきにくい
    - 早めに使いこなせるようになること

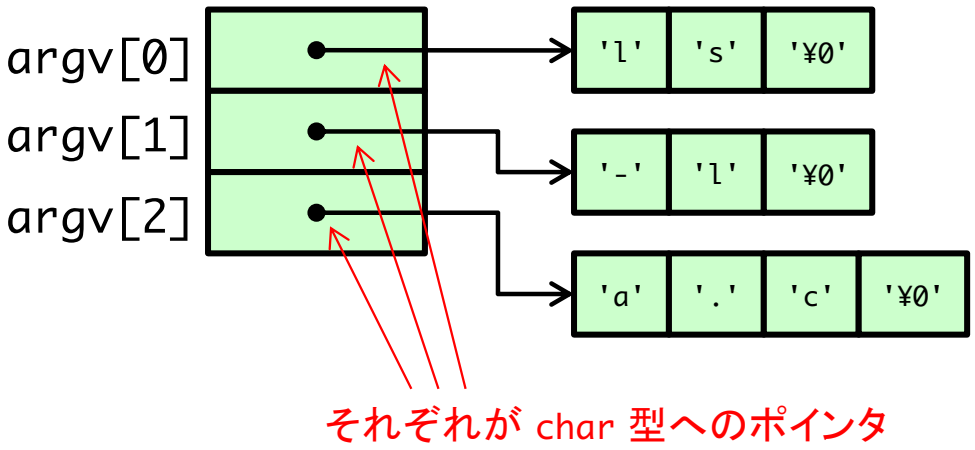
コマンドラインの引数

- UNIX コマンドも C 言語で書かれている
- UNIX コマンドの多くはコマンドライン引数をとる
  - 例: "ls -l a.c"
  - "ls" プログラムの main() はコマンドラインで指定された "-l" や "a.c" を受け取り、処理を行う。
- main() はどのようにしてコマンドライン引数を受け取るのか?

- main() にも引数がある
  - main(int argc, char *argv[])

main(int argc, char *argv[])

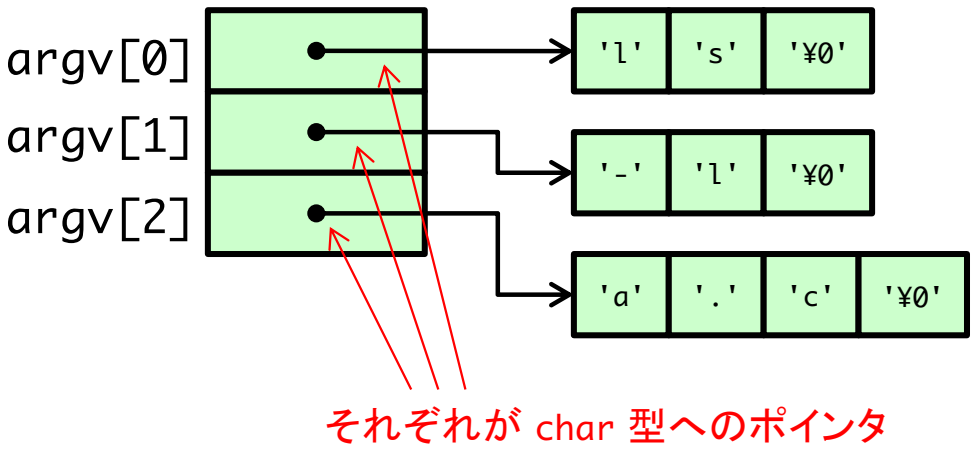
- 慣習的に仮引数名は argc，argv が用いられる。
- argc はコマンド引数の文字列の個数
  - "ls -l a.c" の場合，argc の値は 3 となる。
    - "ls" と "-l" と "a.c" の 3つ
- argv は char 型へのポインタの配列



コマンドライン引数の表示 (1)

```
#include <stdio.h>

main(int argc, char *argv[])
{
    int i;

    for (i = 0; i < argc; i++)
        printf("argv[%d] = %s¥n", i, argv[i]);
}
```

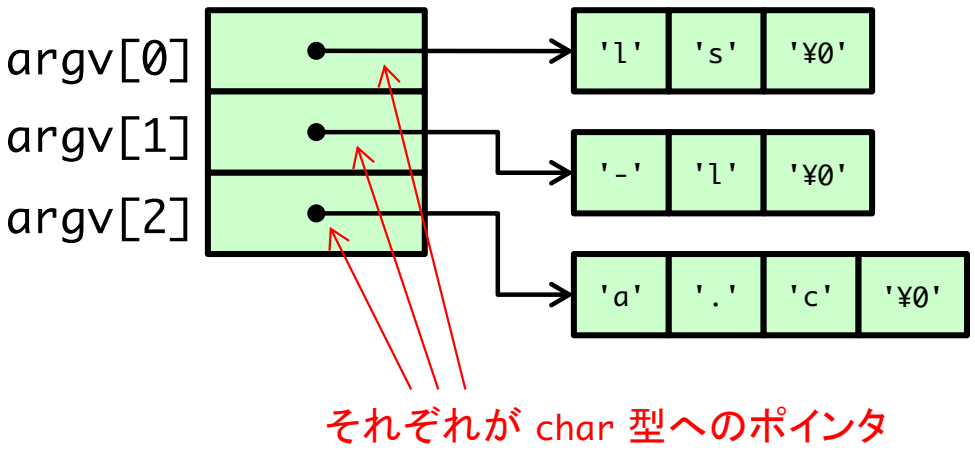
argv を char型への
ポインタの配列として
扱った場合



コマンドライン引数の表示 (2)

```
#include <stdio.h>

main(int argc, char *argv[])
{
    int i;

    for (i = 0; argc > 0; argc--, i++)
        printf("argv[%d] = %s¥n", i, *argv++);
}
```

argv を char 型へのポインタの配列へのポインタとして扱った場合



まとめ

- C 言語におけるファイル入出力の基本
  - `fopen`, `fclose`, `fgetc`, `fputc`, `fgets`, `fputs`
  - `fseek`, `fflush`, `feof`, `ferror` など
- ファイル入出力におけるバッファリング
  - 少し進んだ内容．徐々に理解していけばよい
- 年明けの講義
  - 1/7: 休講
  - 1/12: これまで学んだことを使って画像処理をやってみる
  - 1/19: C 言語の進んだ内容を紹介
    - 演習課題はナシ．TA はいるので質問したり，遅れている課題をやる時間にあてて下さい