情報システムプログラミングⅡ（20回目）

2024年11月6日（水）

3～4限

授業内容

- 講義内容（教科書の548～572ページ＋α）
  - まとまった文字の読み書き
  - サイズ指定による読み書き
  - ランダムアクセス
  - ファイル自体の操作
  - Pythonにおけるファイル入出力
- 演習課題

まとまった文字の読み書き

■fputs関数

- 文字列を書き込む関数

int fputs(const char* dest, FILE* fp);

dest　：ファイルに書き込む文字列

fp　：書き込むファイルのファイルポインタ

戻り値：正常時は正の値、失敗時は EOF (-1)

※文字列の終端までを書き込む (¥0 は含まない)。

この場合は「government of the people,」まで書き込まれる

```
fputs("government of the people, ¥0by the people, ¥0for the people", fp);
```

まとまった文字の読み書き

■fgets関数

- 文字列を読み取る関数

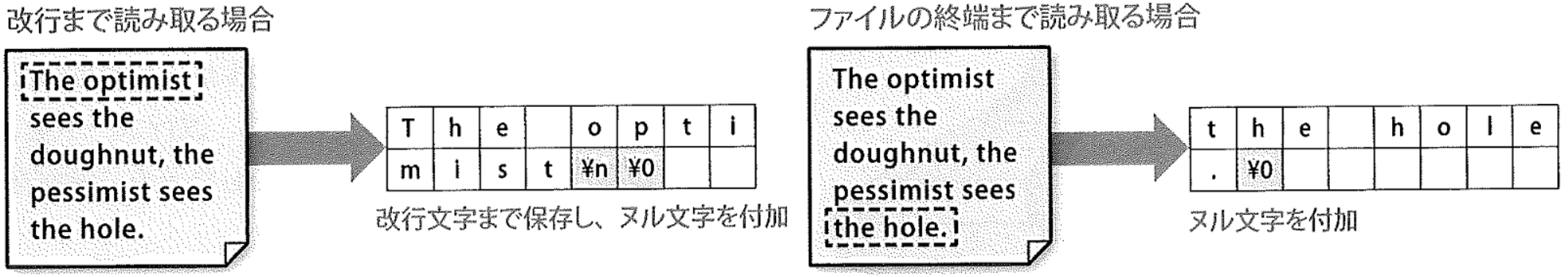
char* fgets(char* dest, int maxlen, FILE* fp)

dest　：読み取った文字列を格納するメモリ領域の先頭アドレス
maxlen：最大読み取りバイト数
fp　：読み取るファイルのファイルポインタ
戻り値：dest、ファイルの終端や失敗時は NULL

※改行文字も文字列として配列に保存する。

改行文字または
ファイルの終端まで
読み取る

確保したメモリ領域を
超えて読み込まないため



まとまった文字の読み書き

■fputs関数とfgets関数の利用例

```
int main(void)
{
  FILE* fp;
  char wbuf[64];

  // 書き込み専用でオープン
  if ((fp = fopen("memo.txt", "w")) == NULL) {
    exit(1);
  }

  fputs("government of the people, ¥nby the people,
    ¥nfor the people", fp);

  fclose(fp);
```

改行文字

文字列を一度に書き込む

```
  // 読み取り専用でオープン
  if ((fp = fopen("memo.txt", "r")) == NULL) {
    exit(1);
  }

  while (fgets(wbuf, 64, fp) != NULL) {
    printf("%s", wbuf);     // 標準出力（画面）に表示
  }

  fclose(fp);

  return 0;
}
```

改行文字までを配列に読み取る

ファイルを最後まで読んだらループ終了

まとまった文字の読み書き

■fprintf関数

- 文字列を書式付きで書き込む関数
  - ➢ ファイルポインタがある以外はprintf関数と同じ書式

int fprintf(FILE* fp, const char* format, …);

fp　　：書き込むファイルのファイルポインタ
format：書式文字列
…　　：書式文字列中のプレースホルダに対応した値
戻り値：書き込んだ文字数、失敗時は負の値

まとまった文字の読み書き

■fprintf関数の利用例

```
typedef struct {
  char   name[16];
  int    height;
  double sight;
} Csv;

int main(void)
{
  FILE* fp;
  char filename[] = "data.csv";

  // データ準備
  Csv data[3] = {
    {"Kaitou", 180, 1.5},
    {"Misaki", 173, 0.6},
    {"Akagi",  161, 1.0}
  };

  if ((fp = fopen(filename, "w")) == NULL) {
    exit(1);
  }
```

```
  for (int i = 0; i < 3; i++) {
    int cn = fprintf(fp, "%s,%d,%4.2f¥n", data[i].name,
        data[i].height, data[i].sight);

    if (cn < 0) {
      printf("書き込みに失敗しました¥n");
      fclose(fp);
      exit(1);
    } else {
      printf("%sさん：%d文字を書き込みました¥n", data[i].name, cn);
    }
  }

  fclose(fp);

  return 0;
}
```

プレースホルダで
書き込みの書式を指定

実行結果

```
Kaitouさん：16文字を書き込みました
Misakiさん：16文字を書き込みました
Akagiさん：15文字を書き込みました
```

```
Kaitou,180,1.50⏎
Misaki,173,0.60⏎
Akagi,161,1.00⏎
```

サイズ指定による読み書き

■バイナリファイルへの読み書き

- fwrite関数：バイト数を指定してファイルに書き込む関数

int fwrite(const void* wp, size_t s, size_t n, FILE* fp)

wp　　：書き込むデータへのポインタ
size　：データ１個あたりのバイト数
n　　：書き込むデータの個数
fp　　：書き込むファイルのファイルポインタ
戻り値：書き込んだデータの個数、失敗時は第３引数よりも小さい値

- fread関数：バイト数を指定してファイルを読み込む関数

int fread(void* rp, size_t s, size_t n, FILE* fp)

rp　　：読み取り領域へのポインタ
size　：データ１個あたりのバイト数
n　　：読み取りデータの個数
fp　　：読み取るファイルのファイルポインタ
戻り値：読み取ったデータの個数、失敗時は第３引数よりも小さい値

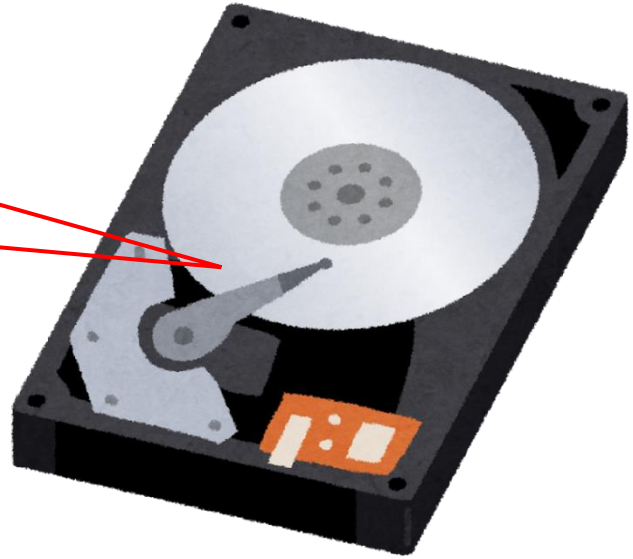
※¥0 は書き込まない。

ランダムアクセス

■ランダムアクセスとは

- 任意の場所に直接アクセスする方式
- 先頭から順にアクセスする方式はシーケンシャルアクセス

目的の場所に針を動かすことを「シーク」，
それに要する時間を「シークタイム」という
（構造が異なるSSDにはシークタイムが無い）



ランダムアクセス

■fseek関数

- 読み書き位置を移動（指定）する関数

int fseek (FILE* fp, long offset, int pos)

fp　　：シーク対象のファイルポインタ
offset　：基準位置から移動するバイト数／ pos：基準位置
戻り値：成功時は 0、失敗時は 0 以外の値

■ftell関数

- 読み書き位置を取得する関数

long ftell(FILE* fp)

fp　　：取得対象のファイルポインタ
戻り値：現在の読み書き位置、失敗時は -1L

ファイル自体の操作

■ファイルの新規作成

- 専用の関数は用意されていない
- fopen関数で書き込みまたは追記を含むアクセスモードを指定することで対応する

■一時ファイルの作成：tmpfile関数

- 一時的に利用可能なファイルを作成可能
  - ファイル名と保存場所は自動で設定され，プログラム終了後は基本的に自動で削除される

FILE* tmpfile()

戻り値：生成したファイルポインタ、失敗時は NULL

ファイル自体の操作

■ファイル名の変更：rename関数

int rename(const char* old, const char* new)

old　　：変更前のファイル名

new　　：変更後のファイル名

戻り値：成功時は 0、失敗時は 0 以外の値

■ファイルの削除：remove関数

int remove(const char* name)

name　：ファイル名

戻り値：成功時は 0、失敗時は 0 以外の値

Pythonにおけるファイル入出力

■open関数：ファイルをオープンする

- 基本的な書式は「f = open(file, mode=' ')」
  - ➢ f：ファイルオブジェクト（名は任意）
  - ➢ file：パスを含むファイル名
  - ➢ mode：モードの設定
    - ✓ r：読み込み（modeの省略時に自動的に設定）
    - ✓ w：書き込み（上書き）
    - ✓ x：ファイルが存在する場合に書き込み（上書き）
    - ✓ a：追記
    - ✓ b：バイナリモード
    - ✓ t：テキストモード（modeの省略時に自動的に設定）
    - ✓ +：モードの拡張（r+やw+など，組み合わせることで読み書き可能に）

「f = open(test.txt, mode='ab'」
など組み合わせ可能

Pythonにおけるファイル入出力

■closeメソッド：ファイルをクローズする

- 基本的な書式は「f.close()」
  - ➢ f：ファイルオブジェクト（オープンした際と同じもの）

■with文：前処理に対する後処理を自動的に行う構文

- ファイル入出力など広く利用される
- ファイル入出力時の基本的な書式は以下の通り

```
with open('test.txt',mode='r') as f:
    print(f.read())
```

ファイルに対して行う処理をブロック内に記述することで,
処理終了後に自動的にクローズされる（closeメソッドが不要に）

Pythonにおけるファイル入出力

■readメソッド：全体を1つの文字列として読み込み

```
with open('test.txt') as f:
    s = f.read()
    print(type(s))
    print(s)
```

type関数により
データの型を
取得可能

modeを省略するとテキスト
モードで読み込む設定となる

■readlinesメソッド：全体を1つのリストとして読み込み

```
with open('test.txt') as f:
    l = f.readlines()
    print(type(l))
    print(l)
```

Pythonにおけるファイル入出力

■readlineメソッド：1行の読み込み

- 右のように反復処理だけでも処理可能

readlineメソッドによる1行のみの読み込み

```
with open('test.txt') as f:
    s_line f.readline()
    print(s_line)
```

readlineメソッド不要で複数行読み込み可能

```
with open('test.txt') as f:
    for s_line in f:
        print(s_line)
```

■writeメソッド：文字列の書き込み

wを指定しているので上書き

```
with open('test.txt',mode='w') as f:
    f.write('TEST')
```

改行文字も書き込まれ，改行として扱われる

```
s = 'TEST¥nTEST'
with open('test.txt',mode='a') as f:
    f.write(s)
```

aを指定しているので追記