#12
ファイルとストリーム
2022 年度 / プログラミング及び実習 III

角川裕次

龍谷大学　先端理工学部

もくじ

今回 (#12) の内容： シラバスでの該当部分

小テーマ: ファイルとストリーム

第 20 回： ファイル入出力の標準ライブラリ関数

重要概念リスト

- ストレージ/ファイルシステム/ファイル
- ストリームの概念
- 標準ストリーム stdin/stdout/stderr
- FILE 型
- 関数：fopen, fclose
- 関数：printf, fprintf, sprintf
- 関数：scanf, fscanf, sscanf
- 関数：time, localtime
- tm 構造体
- バッファオーバーランには要注意

今回の実習・課題 (manaba へ提出)

実習内容と課題内容は講義途中に提示します

(作成したファイル類は manaba に提出)

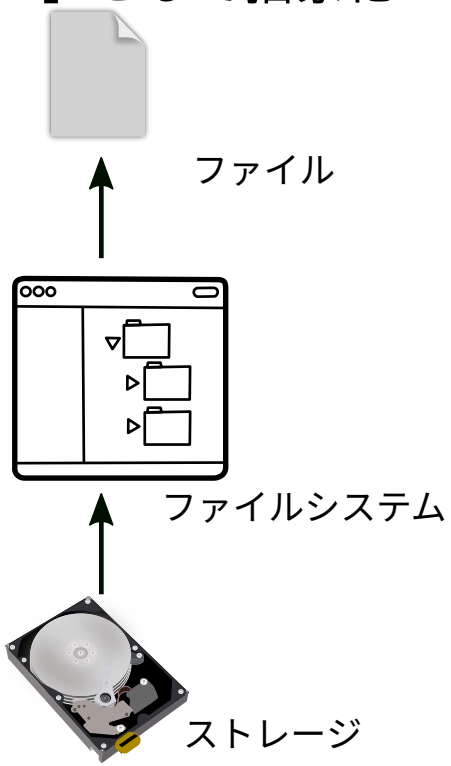
第 13 章 ファイル処理

第 13 章： ファイル関係を学ぶ

ファイル
- データを永続的に記憶する機構
- 永続的： 電源を落としても消えない
- HDD (Hard Disk), SSD (Solid State Drive) などのデバイス上に記憶
- デバイス自体を直接的に扱うのは不便： 「ファイル」として抽象化

ファイルアクセス関数群
- 開く/読む/書く/閉じる/etc.
- ファイル内のデータをストリームとして抽象化

プログラミング技法
- 各種のプログラミングパターン



第 13.1 節　ファイルとストリーム

□ ファイルとストリーム p.352

処理対象のデータや処理結果はストレージへ記憶

- ストレージ：HDD や SSD など
- ファイルとして抽象化

ファイル

- 読み (Read) / 書き (Write) という論理的な機能の観点による抽象化
- ストレージの内部構造は気にしなくてよい

更なる抽象化：キーボードやディスプレイもファイルとみなす

- ストレージ上に書き込む代わりに画面に出力
- ストレージ上のファイルから読み出す代わりにキーボードより入力

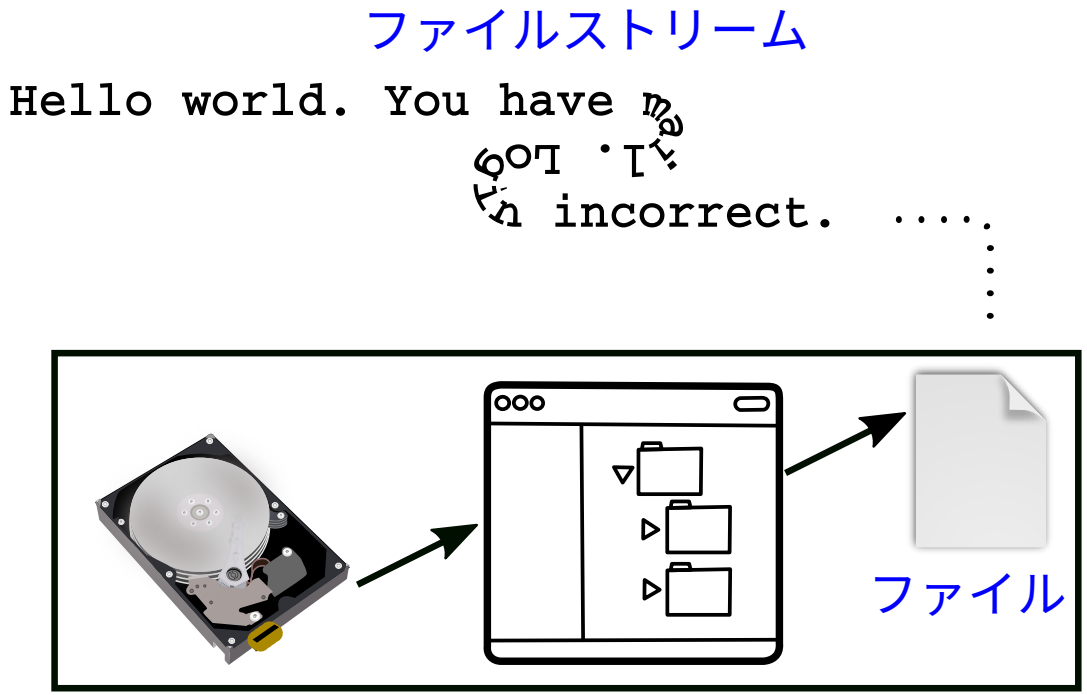
利点：様々な機器を統一的に扱える

ストリーム (stream)

- ファイル・各種デバイスを「文字データの列」とみなしたもの

ストリーム (stream)

ファイル内のデータを文字 (バイト) の流れと抽象化



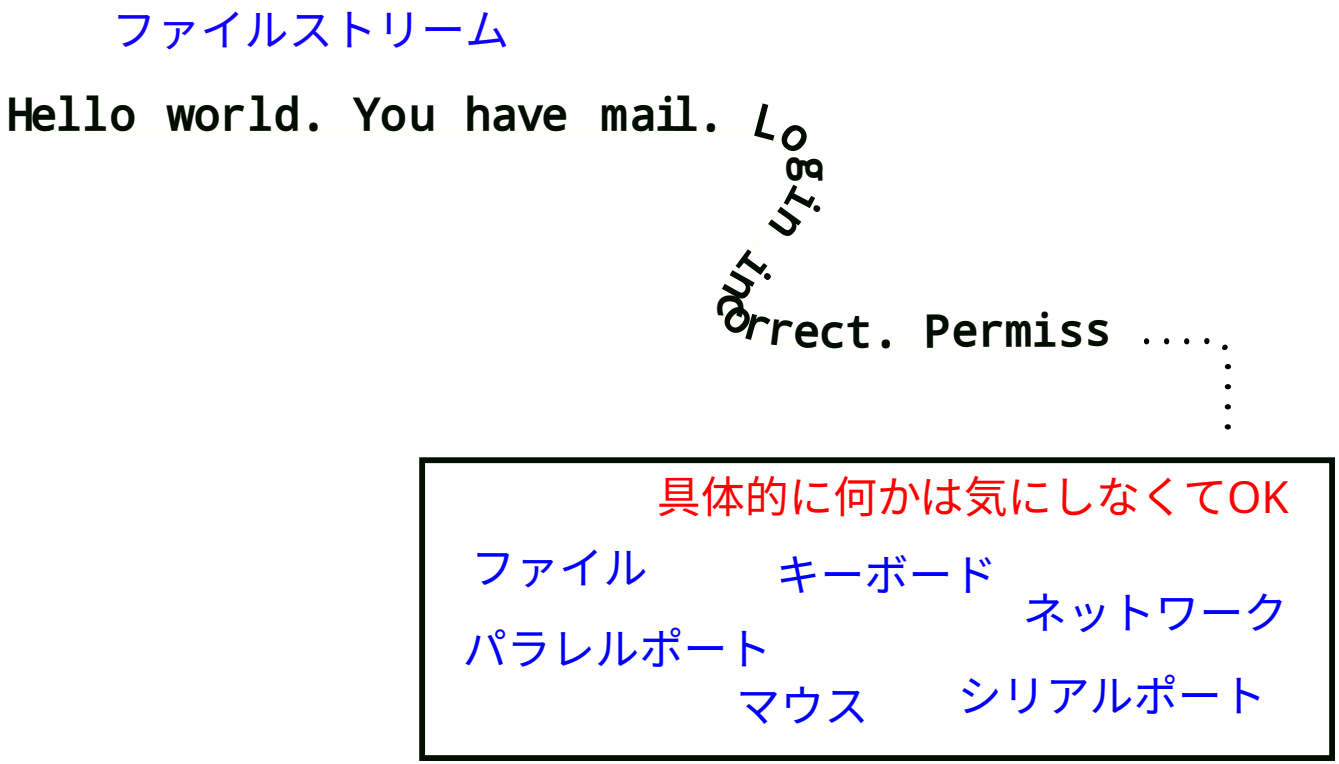
文字が 1 つづつ流れる川 (stream) のようなイメージ

- 読み出し：ファイルの先頭の文字から順番でファイルから読み出す
- 書き込み：ファイルの先頭の文字から順番でファイルへ書き込む

各種デバイスへのアクセスをストリームとして抽象化

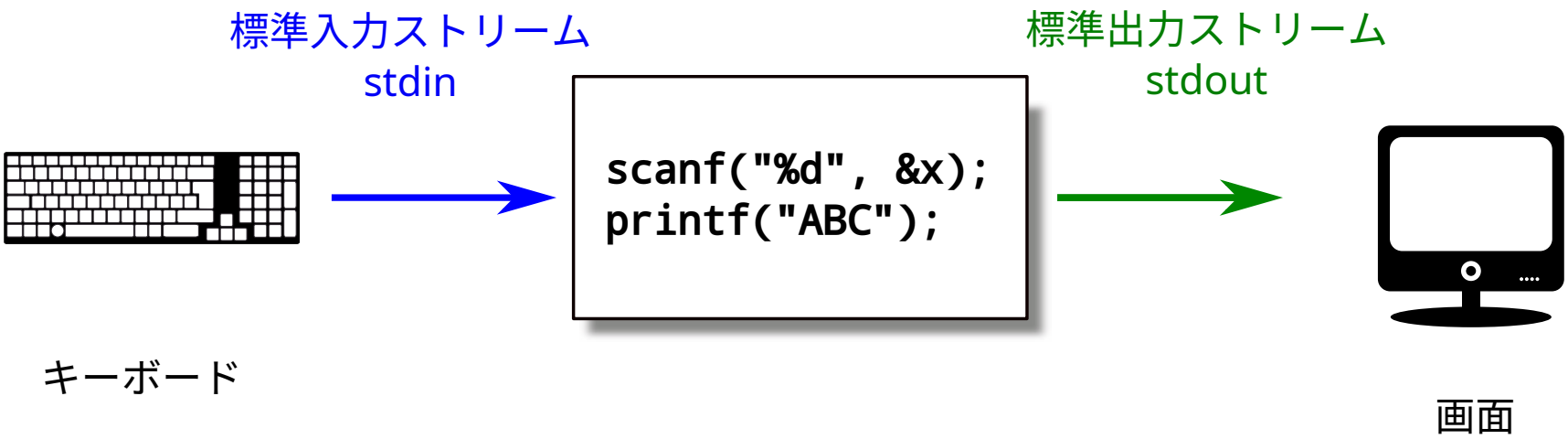
ストリームの先はストレージに限定する必要は特にない
デバイスでも良い

- 文字が 1 つづつ流れてくる川のようなイメージを各種デバイスへ適用



利点：ファイルやデバイスを区別なく統一的にアクセス可能

普段やってる printf と scanf

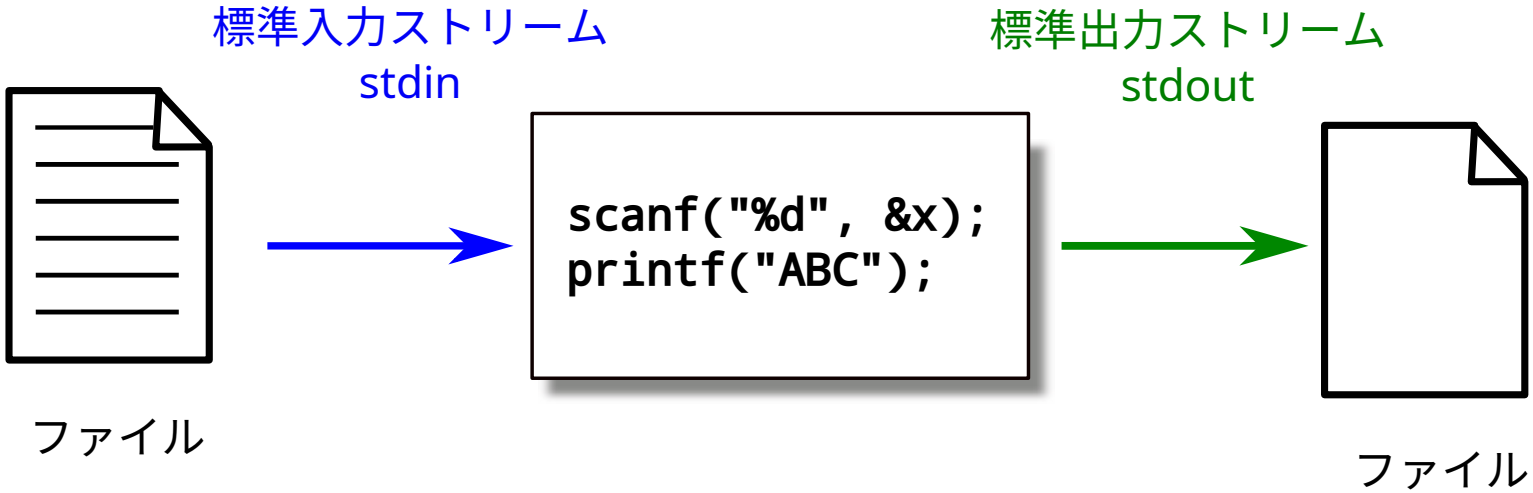


標準的な場合

・入力：キーボードのストリーム

・出力：画面へのストリーム

普段とちがう printf と scanf



入出力先を切り替え可能
・入力：ファイル
・出力：ファイル

このように便利：手作業・目視確認の排除

- キー入力を予めファイルに書いておく
- 出力をファイルに書き込む
- (キーボードも画面も全く関与なし)

★重要★

□ 標準ストリーム　p.353

プログラム実行開始時に 3 種の標準ストリームが事前に準備済み

`stdin` — 標準入力ストリーム (standard input stream)

- 通常の入力を読み取るストリーム
- 標準ではキーボードへ割当
- scanf 関数, getchar 関数などが使用

`stdout` — 標準出力ストリーム (standard output stream)

- 通常の出力を書き込むストリーム
- 標準ではディスプレイ画面へ割当
- printf 関数, puts 関数, putchar 関数などが使用

`stderr` — 標準エラーストリーム (standard error stream)

- エラーメッセージを書き込むストリーム
- 標準ではディスプレイ画面へ割当

第 13.1 節　ファイルのオープンとクローズ

★重要★

□ FILE 型 p.353

構造体 FILE 型

ストリーム制御用のデータ型

- ヘッダ <stdio.h> で定義

以下の情報を含む (直接アクセスは不可)

- (ファイル位置表示子) 現在アクセスしているファイル内での位置
- (エラー表示子) 読み込み/書き込みエラーの記録
- (ファイル終了表示子) ファイルの終りに達したかどうかを記録

※ FILE 型の定義内容は気にしなくて良い

変数の宣言の例 (構造体へのポインタ変数)

```
FILE *fp = NULL; /*通常の使い方*/
```

こういう宣言は普通やらない (構造体変数)

```
FILE fp;   /*普通はこういう使い方はしない*/
```

標準ストリーム

★重要★

標準ストリーム 3 種類用の変数 (FILE*型) が事前に定義済み
(ヘッダ : #include <stdio.h> にて型定義)

stdin
: 標準入力ストリーム

stdout
: 標準出力ストリーム

stderr
: 標準エラーストリーム

★重要★

ファイルをアクセスするプログラムのテンプレ

全体像

```
ファイルを開いてストリームを得る;
ストリームに対して読み書きする;
ファイル(ストリーム)を閉じる;
```

読み出しの場合のテンプレ

```
FILE *fp = fopen(ファイル名, リードモード指定);
if (fp == NULL) {
  エラー (オープン失敗);
}
fpからファイルの内容を読み出して処理を行う;
fclose(fp);
```

書き込みの場合のテンプレ

```
FILE *fp = fopen(ファイル名, ライトモード指定);
if (fp == NULL) {
  エラー (オープン失敗);
}
処理結果を fp へ書き込んでファイル内容を構成する;
fclose(fp);
```

★重要★

□ ファイルのオープン　p.354

ファイルアクセス前に「ファイルをオープン」しストリームを得る

- 以降はストリーム対して読む/書く関数を呼び出す

ファイルをオープン：fopen

関数 `FILE *fopen(const char *filename, const char *mode);`

引数

- `filename`：アクセス対象のファイル名 (文字列)
- `mode`：オープンの際のアクセスモードの指定 (後述)

返却値：オープンしたファイルのストリーム

ヘッダ: `#include <stdio.h>`

例: `fp = fopen("sales.txt", "r");`

fopen オープンモード詳細 (1)

テキストモードを紹介

- プレインテキストの読み書きで使用するモード
- (バイナリモードは後の回で紹介)

`"r"` (読取りモード)

- ファイルが存在しなければエラー
- ファイルの先頭から読み出し

`"r+"` (更新モード)

- ファイルが存在しなければエラー
- ファイルの先頭から読み書き両方が可能

fopen オープンモード詳細 (2)

`"w"` (書込みモード)

- ファイルが存在しなければ新たに作成
- ファイルが存在すれば長さを 0 にする (既存内容をすべて消去)
- ファイルの先頭から書き込み

`"w+"` (更新モード)

- ファイルが存在しなければ新たに作成
- ファイルが存在すれば長さを 0 にする (既存内容をすべて消去)
- ファイルの先頭から読み書き両方が可能

fopen オープンモード詳細 (3)

"a" (追加モード)

- ファイルが存在しなければ新たに作成
- ファイルの終わりの位置から書き込む

"a+" (追加モード)

- ファイルが存在しなければ新たに作成
- ファイルの終わりの位置から書き込む
- 読み出しはファイルの先頭から
- 書き込みはファイルの最後から

★重要★

□ ファイルのクローズ　p.356

読み書きが終わればファイルを「クローズ」を行う (閉じる)

- ストリームの管理情報を解放
- バッファ中の未書き込みデータをフラッシュ (書き出し) も行う

ファイルのクローズ：fclose

関数 `int fclose(FILE *fp);`

引数

- `fp`：ファイルストリーム

返却値：成功すれば 0 (エラー時には EOF)

ヘッダ: `#include <stdio.h>`

例: `fclose(fp);`

□ オープンとクローズの例 p.357

List 13-1 : ファイル abc をオープンしてクローズする例

```
#include <stdio.h>
int main(void)
{
  FILE *fp;
  fp = fopen("abc", "r");
  if (fp == NULL)
    printf("\aファイル\"abc\"をオープンできませんでした。\n");
  else {
    printf("ファイル\"abc\"をオープンしました。\n");
    fclose(fp);
  }
  return 0;
}
```

補足: エラー時には exit(1); で強制終了にするとよい

ファイル読み出し処理のコードパターン

★重要★

ファイル "abc.txt" を読み出しモードでオープン

```
char *fname = "abc.txt";
FILE *fp = fopen(fname, "r");
if (fp == NULL) {
  fprintf(stderr, "%s: not found\n", fname);
  exit(1);
}
  ... fpからファイルの内容を読み出して処理を行う ...
fclose(fp);
```

ファイル書き込み処理のコードパターン

★重要★

ファイル "abc.txt" を書き込みモードでオープン

```
char *fname = "abc.txt";
FILE *fp = fopen(fname, "w");
if (fp == NULL) {
  fprintf(stderr, "%s: failed to open\n", fname);
  exit(1);
}
  ... fpへファイルの内容の書き込みを行う ...
fclose(fp);
```

例: 平方根の表のファイル作成 (要所のみ)

平方根の表をファイル "sqrt.txt" に作成 (ファイルへの書き込み)

```
char *fname = "sqrt.txt";
FILE *fp = fopen(fname, "w");
if (fp == NULL) {
  fprintf(stderr, "%s: failed to create\n", fname);
  exit(1);
}
for (int x = 1; x <= 100; x++) {
  fprintf(fp, "%d %lf\n", x, sqrt(x));
}
fclose(fp);
```

出力 : ファイル sqrt.txt (抜粋) — 各行は x $\sqrt{x}$ の形

```
1 1.000000
2 1.414214
3 1.732051
4 2.000000
5 2.236068
6 2.449490
  ... 略 ...
100 10.00000
```

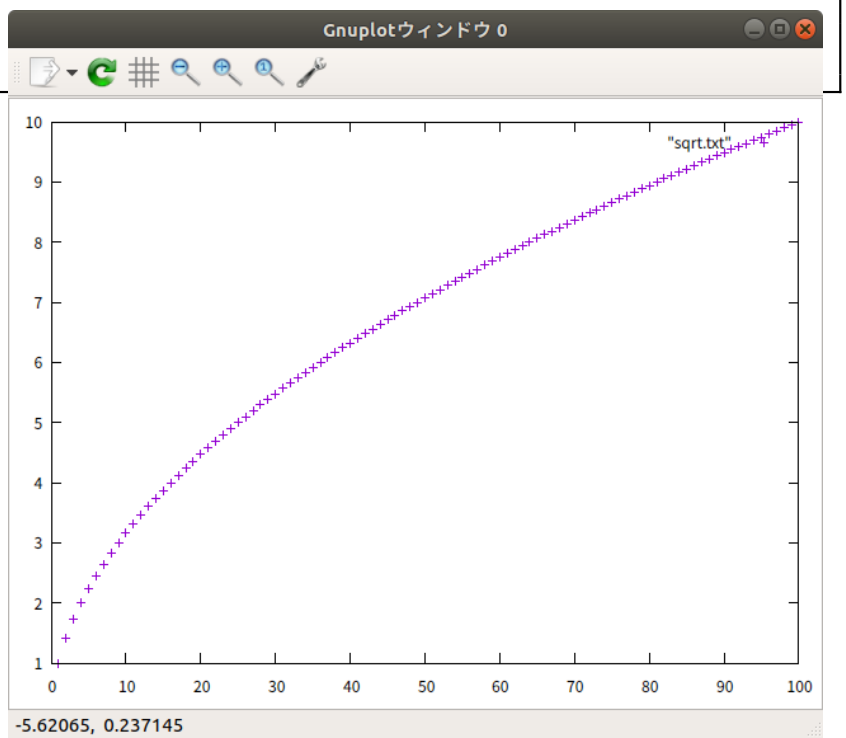
平方根の表：応用

出力ファイルの形式：各行 「x　$f(x)$」

- $f(x)$ は関数値
- 平方根以外の関数でも良い

このファイルを GNUplot に与えればグラフを書いてもらえる

```
$ gnuplot
gnuplot> plot "sqrt.txt"
```



「関数値を計算 → グラフ描く」を C 言語 + GNUplot でできる

「シミュレーション結果を出力 → グラフ描く」もできるよね

□ ファイルのデータの集計 p.358

ファイル中の個人データを集計 (平均値計算)

- 名前, 身長, 体重, の順で各行が構成

```
Aiba 160 59.3
Kurata 162 51.6
Masaki 182 76.5
Tanaka 170 60.7
Tsuji 175 83.9
Washio 175 72.5
```

出力例

```
Aiba          160.0   59.3
Kurata        162.0   51.6
Masaki        182.0   76.5
Tanaka        170.0   60.7
Tsuji         175.0   83.9
Washio        175.0   72.5
----------------------------
平 均         170.7   67.4
```

fscanf 関数によるファイル内容の各行の解析

fscanf: scanf 関数と同等な機能を持つ関数

- ただし指定されたストリーム stream から読む

```
int fscanf(FILE *stream, const char *format, ...);
```

使い方例 1 : `fscanf(fp, "%s%lf%lf", name, &height, &weight);`

使い方例 2 : `fscanf(fp, "%d%d%d", &i, &j, &k);`

使い方例 3 : `fscanf(fp, "%lf%lf", &x, &y);`

scanf と仲間たち

```
int scanf(const char *format, ...);
```

- 標準入力ストリーム stdin を読んで解析

```
int fscanf(FILE *stream, const char *format, ...);
```

- 指定のストリーム stream を読んで解析

```
int sscanf(const char *str, const char *format, ...);
```

- 指定の文字列 str を読んで解析

返却値：実際に読み取ったデータの個数

- ファイル終端の時は EOF が値

返却値の値の一致は必ず検査すること

- ファイルのデータの欠損などのエラーが分かる

scanf 関数の詳細：教科書 358 ページに記載あり

身長・体重を読み込んで平均計算: 全体像

List 13-2 の全体像

```
int main(void)
{
    ファイルをオープンする;
    if (オープン失敗) {
        エラーメッセージ表示;
    } else {
        ファイルを読んで平均値を計算;
        平均値を表示;
        ファイルをクローズ;
    }
}
```

身長・体重を読み込んで平均計算 (1/3)

List 13-2 身長・体重を読み込んで平均を表示 (1/3)

```
#include <stdio.h>
int main(void)
{
  FILE *fp;

  if ((fp = fopen("hw.txt", "r")) == NULL)
    printf("\aファイルをオープンできません。\n");
  else {
...つづく...
```

以下の部分に該当:

```
int main(void)
{
  ファイルをオープンする;
  if (オープン失敗) {
    エラーメッセージ表示;
```

身長・体重を読み込んで平均計算 (2/3)

List 13-2 身長・体重を読み込んで平均を表示 (2/3)

```
    int     ninzu = 0;          // 人数
    char    name[100];          // 名前
    double height, weight;      // 身長・体重
    double hsum = 0.0;          // 身長の合計
    double wsum = 0.0;          // 体重の合計
    while (fscanf(fp, "%s%lf%lf",
                      name, &height, &weight) == 3) {
      printf("%-10s %5.1f %5.1f\n",
             name, height, weight);
      ninzu++;
      hsum += height;
      wsum += weight;
    }
...つづく...
```

以下の部分に該当:

```
    ファイルを読んで平均値を計算;
```

身長・体重を読み込んで平均計算 (3/3)

List 13-2 身長・体重を読み込んで平均を表示 (3/3)

```
    printf("-----------------------\n");
    printf("平均        %5.1f %5.1f\n",
           hsum / ninzu, wsum / ninzu);
    fclose(fp);                          // クローズ
  }
  return 0;
}
```

以下の部分に該当:

```
    平均値を表示;
    ファイルをクローズ;
  }
}
```

重大な問題点が... バッファオーバーラン

List 13-2 : プログラムコード抜粋

```
    char    name[100];        // 名前
      ...
    while (fscanf(fp, "%s%lf%lf",
                          name, &height, &weight) == 3) {
```

バッファオーバーランが発生しますよね...

- 100 を大きな値に変えればいいってものではない

この問題を完全に回避するようコードを修正してみましょう

□ 日付と時刻の書き込み p.360

ファイルに実行時の日付と時刻をファイルに書き込んで保存しておく
(次回に読み出し)

Q. それ, なんの役に立つの?

A1. アプリの設定情報の自動保存
- 次回のアプリ起動時にその情報を読み込む
- 前回の設定情報を引き継げる

A2. しばらく使っていないとパスワードの再入力を要求
- パスワード入力した日時を記録
- 一定期間ごとにパスワードのチェックをしてセキュリティ向上

A3. ファイルシステムの差分バックアップ
- 全体バックアップが保存された日時を記録
- その日時以降に更新されたファイルだけを保存 (差分バックアップ)
- 差分バックアップのサイズは小さい : ストレージ記憶容量の節約

プログラム実行時の日付と時刻をファイルに記録

プログラム構造

1. 現在の時刻情報を得る
2. ファイルを書き込みモードでオープン
3. 既に得た時刻情報をファイルに書く
4. ファイルをクローズ

使用する関数

- time 関数： 現在時刻 (暦時刻; ある時点からの経過秒数) を得る
- localtime 関数： 暦時刻から年月日・時秒を求める
- fopen 関数/fclose 関数： ファイルのオープン/クローズ
- fprintf 関数： 指定のストリームへ書式付きで書き込む

出力例

ファイル dt_dat の内容例

```
2021  5  10  15  29  12
```

(2021 年 5 月 10 日 15 時 29 分 12 秒に実行した場合)

実装例

List 13-3：実行した日付・時刻をファイルに書き出す

```
#include <time.h>
#include <stdio.h>
int main(void)
{
  FILE *fp;
  if ((fp = fopen("dt_dat", "w")) == NULL) // オープン
    printf("\aファイルをオープンできません。\n");
  else {
    time_t current = time(NULL); // 現在の暦時刻
    struct tm *timer = localtime(&current); // 要素別の時刻
    printf("現在の日付・時刻を書き出しました。\n");
    fprintf(fp, "%d %d %d %d %d %d\n",
      timer->tm_year + 1900, timer->tm_mon + 1,
      timer->tm_mday, timer->tm_hour,
      timer->tm_min, timer->tm_sec);
    fclose(fp);                          // クローズ
  }
  return 0;
}
```

time 関数：現在時刻を暦時刻の形で得る

現在時刻を得る

関数 `time_t time(time_t *tloc);`
暦時刻 (1970 年 1 月 1 日からの経過秒数) を返す (関数の返却値)

- UTC 時間で暦時刻を得る
- 引数 `tloc` に非 NULL の値が渡されるとそこへも書き込む

ヘッダ: `#include <time.h>`

UTC：協定世界時（Coordinated Universal Time)

- JST (日本標準時)：UTC に 9 時間加えることで得られる

例

- 例 1：`time(&current);`
- 例 2：`current = time(NULL);`

暦時刻データ型

型 `time_t`

暦時刻を表すデータ型

ヘッダ: `#include <time.h>`

localtime 関数：暦時刻を地方時の年月日・時秒へ解析

暦時間を地方時へ解析

関数 `struct tm *localtime(const time_t *timep);`

動作

- 年月日・時秒に分解
- tm 構造体を返す

例

```
time_t current = time(NULL);
struct tm *t = localtime(&current);
```

tm 構造体：時刻情報を表現

tm 構造体

ヘッダ: `#include <time.h>`

```
struct tm {
   int tm_sec;     /* 秒   [0-60] (1うるう秒) */
   int tm_min;     /* 分   [0-59] */
   int tm_hour;    /* 時   [0-23] */
   int tm_mday;    /* 日   [1-31] */
   int tm_mon;     /* 月   [0-11] */
   int tm_year;    /* 年 - 1900.  */
   int tm_wday;    /* 曜日 [0-6] */
   int tm_yday;    /* 1月1日からの日数 [0-365]  */
   int tm_isdst;   /* 夏時間フラグ [-1/0/1] */
};
```

- tm_mon：0 (1 月), 1 (2 月), ..., 11 (12 月)
- tm_wday：0 (日曜), 1 (月曜), ..., 6 (土曜)
- tm_year：0 (1900 年), 1 (1901 年), ..., 122 (2022 年), ...

time 関数, tm 構造体, localtime 関数/組み合わせ使用例

List 13C-1：実行時の時刻を表示

```
#include <time.h>
#include <stdio.h>
int main(void)
{
  time_t current = time(NULL);    /* 現在の暦時刻 */
  struct tm *timer = localtime(&current); /* 地方時 */
  char *wday_name[] = {"日", "月", "火", "水",
                       "木", "金", "土"};
  printf("現在の日付・時刻は%d年%d月%d日（%s）"
         "%d時%d分%d秒です。\n",
    timer->tm_year + 1900, /* 年（1900を加えて求める） */
    timer->tm_mon + 1,     /* 月（1を加えて求める） */
    timer->tm_mday,        /* 日 */
    wday_name[timer->tm_wday], /* 曜日（0〜6） */
    timer->tm_hour,    /* 時 */
    timer->tm_min,     /* 分 */
    timer->tm_sec      /* 秒 */
  );
  return 0;
}
```

fprintf 関数

printf 関数と類似
オープンしたファイルへの書き込みの際によく使う

書式付き表示関数：fprintf

関数 `int fprintf(FILE *stream, const char *format, ...);`
指定のストリーム `stream` に書式付けで出力

`printf(format, ...);` は
`fprintf(stdout, format, ...);` と等価

- `stdout` はよく使うので書くのを省略できるようにしたもの

□ 前回実行時の情報を取得 p.364

前回の実行日時を表示する機能の実現

- 表示後には今回の実行日時に更新

実行イメージ

- 初めての実行したときの表示

```
本 プ ロ グ ラ ム を 実 行 す る の は 初 め て で す ね 。
```

- 2 回目以降に実行したときの表示

```
前 回 は2021 年5 月1 日23 時42 分19 秒 で し た 。
```

実装方針

実行時の日時を記録するファイルを用意

- ファイル名 `datetime.dat`
- ただし最初はそのファイルは存在しない

ファイル `datetime.dat` の存在を調べる

- ファイルなし：今回が初回の実行
- ファイルあり：今回は 2 回目以降の実行 (ファイル内容を表示)

現在の時刻をファイルに書き込む (ファイル生成)

- 次回の実行で使用

ファイル内の日時データの書式：年 月 日 時 分 秒 (スペース区切り)
例：2021 年 5 月 1 日 23 時 42 分 19 秒の場合

```
2021 5 1 23 42 19
```

実装の詳細 (1/3)

List 13-4 (部分) : ヘッダのインクルードと記録ファイル名

```
#include <time.h>
#include <stdio.h>
char data_file[] = "datetime.dat";   /*実行日時記録用*/
```

List 13-4 (部分) : main 関数

```
int main(void)
{
  get_data(); /* 前回の日付・時刻を取得・表示  */
  put_data(); /* 今回の日付・時刻を書き込む  */
  return 0;
}
```

実装の詳細 (2/3)

List 13-4 (部分)：前回の実行日時の取得と表示

```
void get_data(void)
{
  FILE *fp;

  if ((fp = fopen(data_file, "r")) == NULL) // オープン
    printf("本プログラムを実行するのは初めてですね。\n");
  else {
    int year, month, day, h, m, s;

    fscanf(fp, "%d%d%d%d%d%d",
           &year, &month, &day, &h, &m, &s);
    printf("前回は%d年%d月%d日%d時%d分%d秒でした。\n",
           year, month, day, h, m, s);
    fclose(fp);                          // クローズ
  }
}
```

実装の詳細 (3/3)

List 13-4 (部分) : 現在の日時を書き込む

```
void put_data(void)
{
  FILE *fp;
  if ((fp = fopen(data_file, "w")) == NULL) // オープン
    printf("\aファイルをオープンできません。\n");
  else {
    time_t current = time(NULL); // 現在の暦時刻
    struct tm *timer = localtime(&current); // 要素別の時刻
    fprintf(fp, "%d %d %d %d %d %d\n",
        timer->tm_year + 1900, timer->tm_mon + 1,
        timer->tm_mday, timer->tm_hour,
        timer->tm_min, timer->tm_sec);
    fclose(fp);                          // クローズ
  }
}
```

おわり

番外編の課題 1

指定ファイル内で指定文字列の有無を検索するプログラムの作成

- 指定文字列を含む行を表示
- その行番号も表示

```
$ ./search
文字列: Hello
ファイル名: file.txt
10: Hello, Alice. You have mail.
29: You have mail. Hello world. Login incorrect.
```

10 行目と 29 行目に Hello が含まれている.
他の行には含まれていない