#13
ファイル入出力のプログラミング

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 (#13) の内容：シラバスでの該当部分

小テーマ: ファイル入出力のプログラミング

第 21 回：ファイル入出力のプログラミング

重要概念リスト

- `fgetc` 関数
- `fputc` 関数
- テキストモードとバイナリモードでのファイルアクセス
- `fwrite` 関数と `fread` 関数
- scanf の落とし穴を理解する
- scanf の動作を理解してから適切な使い方をすること

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 13.1 節　ファイルとストリーム (つづき)

□ ファイルの中身の表示 p.366

プログラム構造

```
ファイル名を入力;
fp = ファイルをオープン(読み出しモード);
fp が EOF に到達しない間 {
    fp から読み込んだ1文字を標準出力へ出力;
}
fpをクローズ;
```


プログラムコード

List 13-5 : ファイルの中身を表示

```
#include <stdio.h>
int main(void)
{
  FILE *fp;
  char fname[FILENAME_MAX];  // ファイル名
  printf("ファイル名:");
  scanf("%s", fname);
  if ((fp = fopen(fname, "r")) == NULL) // オープン
    printf("\aファイルをオープンできません。\n");
  else {
    int ch;
    while ((ch = fgetc(fp)) != EOF)
      putchar(ch);
    fclose(fp);                         // クローズ
  }
  return 0;
}
```


ストリームからの 1 文字入力

ストリームから 1 文字入力 : fgetc

関数 `int fgetc(FILE *stream);`

引数
- stream : 対象ストリーム

返却値
- 読んだ文字
- ファイルの終了時には EOF

ヘッダ : `#include <stdio.h>`

中心部分

ファイルオープン: お決まりのパターン

```
if ((fp = fopen(fname, "r")) == NULL) // オープン
  printf("\aファイルをオープンできません。\n");
```

読み込み: お決まりのパターン

```
int ch;
while ((ch = fgetc(fp)) != EOF)
  putchar(ch);
fclose(fp);
```


マクロ記号 FILENAME_MAX

マクロ記号 FILENAME_MAX

使用の処理系におけるファイル名の最大長
- ヘッダ<stdio.h>で定義 (オブジェクト形式のマクロ)

注意 : 処理系や OS が違うと具体的な値は異なる場合あり
- あるシステムでは 4096 と定義されている
- 別のシステムでは 1024 と定義かも
- 具体的な値に依存したコードは書かないこと
- 数値の直打ちは移植性/ポータビリティのないソースコード

結論 : マクロ定義された記号を使用すべき

□ ファイルのコピー p.368

プログラム構造

```
コピー元のファイル名を入力;
コピー先のファイル名を入力;

sfp = コピー元のファイルをオープン(読み出しモード);
dfp = コピー先のファイルをオープン(書き込みモード);

sfp が EOF に到達しない間 {
    sfp から読み込んだ1文字を dfp へ出力;
}

dfpをクローズ;
sfpをクローズ;
```

プログラムコード全体

List 13-6: ファイルをコピーする

```
#include <stdio.h>
int main(void)
{
  FILE *sfp;            // コピー元ファイル
  FILE *dfp;            // コピー先ファイル
  char sname[FILENAME_MAX];    // コピー元のファイル名
  char dname[FILENAME_MAX];    // コピー先のファイル名
  printf("コピー元ファイル名:");   scanf("%s", sname);
  printf("コピー先ファイル名:");   scanf("%s", dname);
  if ((sfp = fopen(sname, "r")) == NULL) // コピー元をオープン
    printf("\aコピー元ファイルをオープンできません。\n");
  else {
    if ((dfp = fopen(dname, "w")) == NULL) // コピー先をオープン
      printf("\aコピー先ファイルをオープンできません。\n");
    else {
      int ch;
      while ((ch = fgetc(sfp)) != EOF)
        fputc(ch, dfp);
      fclose(dfp);            // コピー先をクローズ
    }
    fclose(sfp);              // コピー元をクローズ
  }
  return 0;
}
```


プログラムコード詳細 (1/3)

List 13-6 (1/3): 変数 & ファイル名

```
#include <stdio.h>
int main(void)
{
  FILE *sfp;            // コピー元ファイル
  FILE *dfp;            // コピー先ファイル
  char sname[FILENAME_MAX];    // コピー元のファイル名
  char dname[FILENAME_MAX];    // コピー先のファイル名
  printf("コピー元ファイル名:");   scanf("%s", sname);
  printf("コピー先ファイル名:");   scanf("%s", dname);
```

プログラム構造の以下の部分に該当

```
コピー元のファイル名を入力;
コピー先のファイル名を入力;
```


プログラムコード詳細 (2/3)

List 13-6 (2/3): ファイルオープン

```
  if ((sfp = fopen(sname, "r")) == NULL) // コピー元をオープ
    printf("\aコピー元ファイルをオープンできません。\n");
  else {
    if ((dfp = fopen(dname, "w")) == NULL) // コピー先をオー
      printf("\aコピー先ファイルをオープンできません。\n");
```

プログラム構造の以下の部分に該当

```
sfp = コピー元のファイルをオープン(読み出しモード);
dfp = コピー先のファイルをオープン(書き込みモード);
```


プログラムコード詳細 (3/3)

List 13-6 (3/3): コピー & ファイルクローズ

```
    else {
      int ch;
      while ((ch = fgetc(sfp)) != EOF)
        fputc(ch, dfp);
      fclose(dfp);              // コピー先をクローズ
    }
    fclose(sfp);                // コピー元をクローズ
```

プログラム構造の以下の部分に該当

```
sfp が EOF に到達しない間 {
    sfp から読み込んだ1文字を dfp へ出力;
}
dfpをクローズ;
sfpをクローズ;
```


ストリームへの 1 文字出力

ストリームへ 1 文字を書き込む : fputc

関数 `int fputc(int c, FILE *stream);`

引数

- c : 書き込む文字
- stream : 書き込み先ストリーム

返却値

- 書き込んだ文字
- エラー時には EOF

ヘッダ : `#include <stdio.h>`

中心部分

お決まりのパターン: あるストリームから別のストリームへコピー

```
    while ((ch = fgetc(sfp)) != EOF)
      fputc(ch, dfp);
```

やってること

1. 1 文字をストリーム sfp から読む (fgetc 関数)
2. ストリームの終わり (EOF) なら終了
3. 読んだ 1 文字を dfp へ書き込み (fputc 関数)
4. 繰り返す

第 13.2 節　テキストとバイナリ

第 13.2 節で学ぶこと：バイナリファイルの取扱

★重要★

テキストファイルとは?

- ファイル内容が文字情報だけで構成
  (改行やタブも文字情報の範疇)
- いわゆる「プレインテキスト (plain text)」
- Windows「メモ帳」で作るファイルはテキストファイル
- 本講義でこれまでに扱ったファイルはテキストファイル

バイナリファイルとは?

- 文字以外の情報 (可読文字ではないデータ) を含む
- 例：PDF, JPEG, WAV, MP3, Word, Excel, PowerPoint, ZIP など

注意：Word ファイル (.docx) はバイナリファイル

- 文字だけの Word 文書であってもテキストファイルではない

バイナリファイルが扱えるようになると嬉しいこと

★重要★

・音声ファイル を加工/解析するプログラムを自作できる
- WAV, MP3 等

・画像ファイルを加工/解析するプログラムを自作できる
- JPEG, GIF, PNG 等

・画面でグラフィックスを描画するプログラムを自作できる
- キャラクターやイメージ画像の描画

・組込みシステム等でのハードウエア制御のプログラムを自作できる
- センサーの測定値データの読み取り
- 赤外線リモコンの通信プロトコル実装

・デジタル・フォレンジック技術
- ストレージ上のファイル残骸の解析・消失データの復元

□ テキストファイルへの実数値の保存　p.370

ファイルに実数値を保存する方法のひとつ

- `fprintf(fp, "%f", v);` 等の形でファイルへ書き込み
- テキストエディタ等でファイルを開いたとき人間に認識容易な文字列
- `fscanf` をつかえばコンピュータ内に数値の読み込みが可能

詳しいことは省略

バイナリファイルの用途
　実数値のファイル保存よりも
　アプリケーション固有のファイルの読み書きで重要

- 画像ファイル, 動画ファイル, 音声ファイル, ワープロ文書ファイル, 表計算ファイル, などはバイナリファイル
- アプリケーションプログラム内部ではバイナリファイルの読み書き

□ テキストファイルとバイナリファイル　p.371

★重要★

データをどのようなデータ形式でファイルに保存するか: 2 通り

- テキストファイル
- バイナリファイル

テキストファイル

- 人間可読 (human-readble) な文字の列で表現
- コンピュータが読み込むには 2 進数表現への変換が必要
- 人間がテキストエディタでファイルを開いて読み書き編集が可能

バイナリファイル

- コンピュータ内部の表現 (2 進数) のままのバイト列で表現
- 人間が理解するには 2 進数の表現の解釈が必要
- 専用アプリケーションソフトウエアの利用が必要

テキストファイルとバイナリファイルの例

★重要★

int 型変数 n をテキスト/バイナリそれぞれの方法で出力

```
int n = 357;
```

テキストファイルとして出力：

```
fprintf(fp1, "%d", n);
```

- 整数値を ASCII 文字列に変換

出力の 16 進数ダンプ

```
33 35 37
```

- 数字列が ASCII コードで表示
  0x33='3', 0x35='5',
  0x37='7'

バイナリファイルとして出力：

```
fprintf(fp2, "%c%c",
        (n>>8) & 0xFF,
        n & 0xFF);
```

- 整数値の 2 進数表現を上位 8 ビット, 下位 8 ビットの順で

出力の 16 進数ダンプ

```
01 65
```

- 2 バイトが 16 進数で表示
  0x0165 = 357

出力書式 %c に関する注意

書式指定 %c の意味：指定された数値を 1 バイトで出力

コード例

```
fprintf(fp, "%c", n);
```

書式指定は %c だが文字 (テキスト) で出力されるわけではない
- 出力として与える値で決まる

n=0x31 の場合：0x31 (可読文字, 数字の'1') が出力

n=1 の場合：0x01 (非可読文字) が出力

□ バイナリファイルへの実数値の保存 p.372

実数値をコンピュータの内部表現のままファイルに保存するには?
- バイナリ (2 進数) 表現のままで
- 人間にとっての可読形式 (10 進数文字列) に変換するのではない

用いる関数には 2 通りあり

1. fprintf 関数の %c 書式
   - (複数バイトで表現されている) 値を 1 バイトづつに分解して出力
   - 長所：様々な CPU で共通利用できるデータファイルを構成できる
   - 短所：1 バイトづつへの分解が必要
2. fwrite 関数と fread 関数
   - 複数バイトを一度に読み書きする
   - 長所：コードは簡単
   - 短所：データファイルは保存を行った CPU でしか読み出せない

バイナリ出力に便利な fwrite 関数

ストリームへの書き込み：fwrite

関数 `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);`

引数
- ptr：書き込み内容のアドレスから
- stream：書き込み先ストリーム
- nmemb：書き込む要素の個数
- size：1 要素のバイト数

返却値
- 書き込みに成功した要素数

ptr から始まる合計 nmemb×size バイトを順に出力
- メモリの内容をそのままバイナリ形式で出力

fwrite 関数の使用例

```
#define N  100
double a[N];
fwrite(a, sizeof(double), N, fp_out);
```


バイナリ入力に便利な fread 関数

ストリームからの読み出し：fread

関数 `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`

引数
- ptr：読み出し内容の格納先
- size：1 要素のバイト数
- nmemb：読み出す要素の個数
- stream：読み出しストリーム

返却値
- 読み込みに成功した要素数

合計 nmemb×size バイトを ptr から書き込む
- 読み取り内容をそのままバイナリ形式でメモリへ書き込む

fread 関数の使用例

例 (コード断片)

```
#define N  100
double b[N];
fread(b, sizeof(double), N, fp_in);
```

バイナリファイルのオープン

fopen 関数のモード指定にバイナリ指定

ファイルのオープン：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
  - `"rb"`　バイナリ読取りモード
  - `"wb"`　バイナリ書込みモード
  - `"ab"`　バイナリ追加モード

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>`

バイナリファイルのオープン例

```
#define FNAME  "saved-data.dat"
#define N  100
double a[N];
FILE *fp_save = fopen(FNAME, "wb");
fwrite(a, sizeof(double), N, fp_save);
```


構造体のまるごと保存も可能

ファイルへ保存する例 (コード断片)

```
#define N  100
#define FNAME  "saved.dat"

struct xyz {
  int x;
  double y;
  char z;
};
struct xyz v[N];

FILE *fp_save = fopen(FNAME, "wb");
fwrite(v, sizeof(struct xyz), N, fp_save);
```

ファイルから読み込むコードもすぐ書けますよね...

ここまで理解できればあとは自力で WAV ファイル解析や WAV ファイル生成プログラムを自作できますね (WAV フォーマットは自分でお勉強して下さい)

□ ファイルのダンプ　p.374

「ダンプ」＝「ファイル内容を 1 バイトづつ表示・出力」

WAV ファイルの 16 進ダンプ

- 表示の左側：アドレス
- 表示の中側：ファイル内容を 16 進数で (16 バイトごと)
- 表示の右側：ASCII 可読文字ならその文字を表示 (非可読なら「.」)

```
$ ./dump
ファイル名：music.wav
00000000 52 49 46 46 77 64 01 00 57 41 56 45 66 6D 74 20 RIFFwd..WAVEfmt
00000010 12 00 00 00 01 00 01 00 40 1F 00 00 40 1F 00 00 ........@...@...
00000020 01 00 08 00 00 00 64 61 74 61 CB 59 01 00 82 7D ......data.Y...}
00000030 98 B4 88 4E 33 1C 68 E8 F8 BD 62 29 0B 6D DA CC ...N3.h...b).m..
00000040 9F 72 4F 21 70 B9 9E 8A 89 5A 2A 85 BB 93 94 96 .rO!p....Z*.....
00000050 4C 32 8D A8 88 98 8B 38 4E 9E 89 70 92 8B 54 8E L2.....8N..p..T.
00000060 C2 87 5B 74 6D 4A A2 C7 74 57 81 64 52 C1 C0 67 ..[tmJ..tW.dR..g
   ... 略 ...
```

これを見れば以下のことが分かるんですよ

- WAV ファイル, サンプリング周波数 8kHz, チャンネル数 1,
  量子化数 8 ビット, PCM コードのバイト数, ... など

ダンプ; プログラムコード

List 13-9 (1/2)：ファイルをオープンするところまで

```
#include <ctype.h>
#include <stdio.h>
int main(void)
{
  FILE *fp;
  char fname[FILENAME_MAX];       // ファイル名
  printf("ファイル名：");
  scanf("%s", fname);
  if ((fp = fopen(fname, "rb")) == NULL) // オープン
    printf("\aファイルをオープンできません。\n");
  else {
```

(つづく)

ダンプ; プログラムコード (つづき)

List 13-9 (2/2)：16 進ダンプ本体

```
    unsigned long count = 0;
    unsigned char buf[16];
    while ((n = fread(buf, 1, 16, fp)) > 0) {
      printf("%08lX ", count);              // アドレス
      for (int i = 0; i < n; i++)           // 16進数
        printf("%02X ", (unsigned)buf[i]);
      if (n < 16)
        for (int i = n; i < 16; i++)
          printf("   ");
      for (int i = 0; i < n; i++)           // 文字
        putchar(isprint(buf[i]) ? buf[i] : '.');
      putchar('\n');
      count += 16;
    }
    fclose(fp);                             // クローズ
  }
  return 0;
}
```

第 13.3 節　printf 関数と scanf 関数

□ printf 関数：書式付きの出力　p.376

書式付きの出力

関数 `int printf(const char *format, ...);`

引数
- format：書式指定
- 以降の引数：書式指定に従い標準出力ストリームへ出力

返却値
- 出力した文字数 (エラー時は負の値)

ヘッダ：`#include <stdio.h>`

printf, fprintf, sprintf：書式指定方法は同じ (書き込み先だけの違い)

書式中の指令：%で始まる記号列で表示形式を指定
- 表示幅, 右寄せ/左寄せ, 小数以下の桁数, などの機能

printf 関数の使用例

```
printf("Sum=%d\n", s);
printf("Sum=%f, Average=%f\n", sum, avr);
printf("A=%f%%, B=%f%%\n", percent_a, percent_b);
```


printf 関数：書式指定での指令 (1/2)

指令：%で始まる以下の要素の列
- フラグ (省略可)
- 最小フィールド幅 (省略可)
- 精度 (省略可)
- 変換修飾子 (省略可)
- 変換指定子

変換指定子 (よく使うものだけ紹介)
- d
  int 型の実引数を 10 進数表記に変換
  例：`"Age=%d"` → `Age=21`
- x
  unsigned 型の実引数を 16 進数表記に変換 (小文字)
  例：`"addr=0x%x\n"` → `addr=0x12fc`
- X
  unsigned 型の実引数を 16 進数表記に変換 (大文字)
  例：`"ADDR=0X%X, DATA=0X%X"` → `ADDR=0X12FC, DATA=0X1A`

printf 関数：書式指定での指令 (2/2)

- f
  double 型の実引数を *ddd.ddd* 形式の 10 進数表記に変換
  例：`"sum=%f"` → `sum=53.128`
- e
  double 型の実引数を *d.ddde+dd* 形式の 10 進数表記に変換
  例：`"C=%e"` → `C=2.99792e+08`
- c
  int 型の実引数を unsigned char 型の文字に変換
  例：`"ch=%c"` → `ch=H`
- s
  char*型の実引数 (ポインタ) で始まる文字列に変換
  例：`"name: %s"` → `name: Bill Brown`
- P
  void*型の実引数のポインタ値を表示可能な文字列に変換
  例：`"object addr: %P"` → `object addr: 0x800ac110`
- %
  %に変換

書式指定の文字列の例 (1/2)

int 型：コード

```
printf("今日は%02d月%02d日\n", 3, 9); /*2桁で(0を補う)*/
printf("今日は% 2d月% 2d日\n", 3, 9); /*2桁で(スペースを補う)
```

実行結果

```
今日は03月09日
今日は  3月  9日
```

double 型

```
printf("総和は%10.2f\n", 123.4567); /*全体10文字, 小数部2桁*/
printf("総和は%10f\n",    123.4567); /*全体10文字*/
printf("総和は%.1f\n",    123.4567); /*小数部1桁*/
```

実行結果

```
総和は    123.46
総和は123.456700
総和は123.5              四捨五入されていることに注意
```

書式指定の文字列の例 (2/2)

文字列 char* : コード

```
printf("M=%s!\n",     "Hello");
printf("M=%10s!\n",   "Hello"); /*全体10文字, 右寄せ*/
printf("M=%-10s!\n", "Hello"); /*全体10文字, 左寄せ*/
```

実行例

```
M=Hello!
M=      Hello!
M=Hello     !
```


□ scanf 関数 : 書式付きの入力 p.380

書式付きの入力 : scanf

関数 `int scanf(const char *format, ...);`
- 標準出力ストリームを書式に従って入力
- 読み込み内容をその後に続くオブジェクトへ格納

引数
- `format` : 読み込みの書式

返却値
- 読み込んだオブジェクト数 (エラー時は負の値)

ヘッダ : `#include <stdio.h>`

scanf, fscanf, sscanf : 書式指定方法は同じ (読み込み元だけの違い)

書式中の指令 : %で始まる記号列で読み取り形式を指定
- 基数, 値を格納するオブジェクトのデータ型, などを指定

scanf 関数の使用例

- `scanf("%d", &s);`
- `scanf("%lf%lf", &sum, &avr);`
- `scanf("%f", percent);`

scanf 関数 : 書式指定での指令 (1/2)

指令 : %で始まる以下の要素の列
- 代入抑制文字* (省略可)
- 最大フィールド幅 (省略可)
- 変換修飾子 (省略可)
- 変換指定子

変換修飾子 (よく使うものだけ紹介)
- `l`
  整数への変換 (`d`, `i`, `n`) の場合は `long` 型
  実数への変換 (`e`, `f`, `g`) の場合は `double` 型 (無指定だと `float` 型)

変換指定子 (よく使うものだけ紹介)
- `d`
  10 進数表記を整数に変換
- `i`
  (8,10,16 進数表記を) 整数に変換

scanf 関数 : 書式指定での指令 (2/2)

- `x` または `X`
  16 進数表記を整数に変換
- `e`, `E`, `f`, `g`, `G`
  浮動小数点数に変換
- `c`
  1 文字を得る
  (フィールド幅が指定されていれば得る文字はその数とする)
- `s`
  非空白文字の並びを得る
- `n`
  入力を読み取らず, これまでに読み取った文字数を得る
- `%`
  %と照合

気づきましたか?
printf と scanf での double/float 型の違い

Q. 指定すべき書式文字列が微妙に非対称なのは一体なんなの???

	double 型	float 型
printf	`printf("%f", &d)`	`printf("%f", &f)`
scanf	`scanf("%lf", &d)`	`scanf("%f", &f)`

A1. `printf` : 実引数の型変換が理由
- `float` 型実引数は `double` 型に型変換されて `printf` 関数を呼び出す
- `printf` 関数が受け取るのは `double` 型
- `printf` 関数は元が `float` 型か `double` 型か区別できない
- 区別できないのだから違う書式文字列を使うことに意味がない

A2. `scanf` : `sizeof(float)` と `sizeof(double)` の違い
- オブジェクトサイズが違うため scanf では区別する必要がある
- 型変換などでは区別はなくせない

scanf 関数を気楽に使うのはおすすめしない (1/3)

★重要★

みなさん scanf を気軽に使ってるようですが... scanf には落とし穴が多い (ほんとうに理解してますか?)

1. ありがちなコード (文字列)

```
char s[20];
print("Name=");
scanf("%s", s);
```

- Bill Brown を入力しても Bill だけを読み取り
- スペースが区切り
- Brown は次の scanf で読まれる

2. ありがちなコード (整数)

```
int d = 0;
print("Value=");
scanf("%d", &d);
```

- 非数字文字 (例:ABC) を入力すると読み取り失敗
- 非数字文字は入力ストリームに残ったままで次の scanf で読まれる
- 次の scanf も整数入力ならそこでも読み取りが必ず失敗

scanf 関数を気楽に使うのはおすすめしない (2/3)

Q. じゃあどうすればいいの?

A1. fgets, atoi, atof の組み合わせ (整数・実数の場合)

- fgets : 1 行を文字列で入力
  - バッファサイズの指定でバッファオーバーランを回避
  - 期待外の文字が入力ストリームに残ることを回避
- atoi : 文字列を整数値に変換
- atof : 文字列を実数値に変換

A2. fgets と sscanf の組み合わせ (文字列でも OK)

- sscanf : (ストリームでなく) 文字列に対して読み取り
- 期待外の文字が入力ストリームに残ることを回避

scanf 関数を気楽に使うのはおすすめしない (3/3)

対話入力のコードを正しく書くのはかなり難しい

- 入力ミスを完全に検出するコードは結構複雑
- 入力ミスを検出できないとおかしな動作を発生

scanf は入力エラーチェックが困難なので使うのはあまりすすめない

それでも scanf を使うなら以下の Web ページなどを読んで理解してから

「悪名高く，恐ろしい scanf() 関数」,
https://qiita.com/epppJones/items/682dbc02a3e1b15846af
(2020 年 11 月 14 日閲覧)

scanf の代わりのコード (数値)

整数値を変数 n に読む

scanf を使ったコード

```
char s[80];
int n;
if (scanf("%d", &n)
    != 1) {
  exit(1);
}
```

読み取ったデータ数を
チェック

scanf を使わないコード

```
char s[80];
int n;
if (fgets(s, sizeof(s), stdin)
    == NULL) {
  exit(1);
}
if (sscanf(s, "%d", &n) != 1) {
  exit(1);
}
```

行末の Enter キーによる
\n (改行文字) を削除

読み取ったデータ数をチェック

scanf の代わりのコード (文字列)

文字列を読み込む

scanf を使ったコード

```
char s[80];
if (scanf("%s", s)
    != 1) {
  exit(1);
}
```

読み取ったデータ数を
チェック

入力 Bill Brown に対して
Bill しか読めない

scanf を使わないコード

```
char s[80];
if (fgets(s, sizeof(s), stdin)
    == NULL) {
  exit(1);
}
char *p = strchr(s, '\n');
if (p != NULL){
  *p = '\0';
}
```

行末の Enter キーによる
\n (改行文字) を削除

入力 Bill Brown すべてを読める

おわり

番外編の課題 1

指定された WAV ファイルの情報を表示するプログラムの作成

- WAV フォーマットは各自で調査

表示すべき項目

- サンプリング周波数
- チャンネル数
- 量子化数

```
$ ./wavinfo
ファイル名: music.wav
サンプリング周波数: 44.1kHz
チャンネル数: 2
量子化数: 16
```