プログラミングI 第
数理物理，総合理学等向け

2018年11月24日 土

海谷 治彦

目次

- ファイルとは何か？
- プログラムからファイルを扱うには？
- ファイルの読み込み
- ファイルの書き込み
- ファイルの削除，名称変更

電源を切ってもデータが残ってる！

- コンピュータの電源を切って再起動しても，データは残っている.
  - ワードの文書，MP3ファイル，画像ファイル，プログラムファイル・・・・
- これは，コンピュータの中でデータを保管してくれているからである.
- データを保管する単位をファイルと呼び，保管庫をファイルシステムと呼ぶ.
  - ファイルシステムはOSの一部であることが多い.
- ファイルシステムでは，特定のファイル群をまとめて保管する機能がある.
- このまとめる部分をフォルダやディレクトリと呼ぶ.

オペレーティング システム

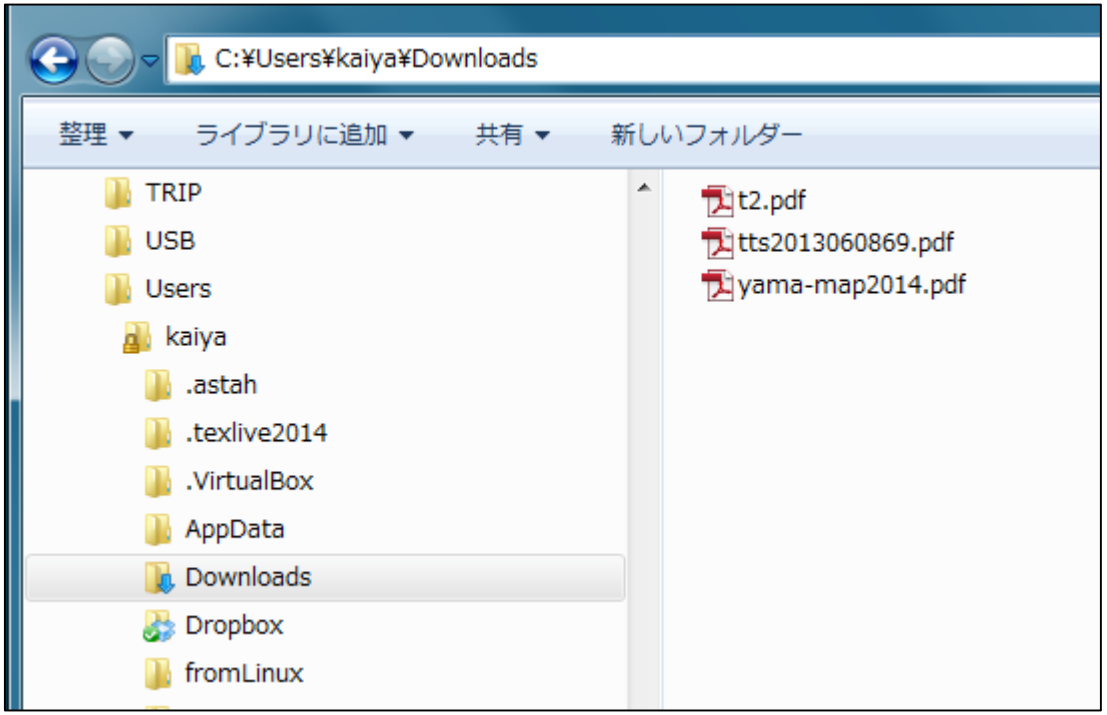
Operating System (OS)

- Windows 10, 8, 7, XP ...
- UNIX
- Linux
- OS X (MacOS)
- Android
- iOS
- .....

興味がある人は2年前期に
「オペレーティングシステム」の授業あります.

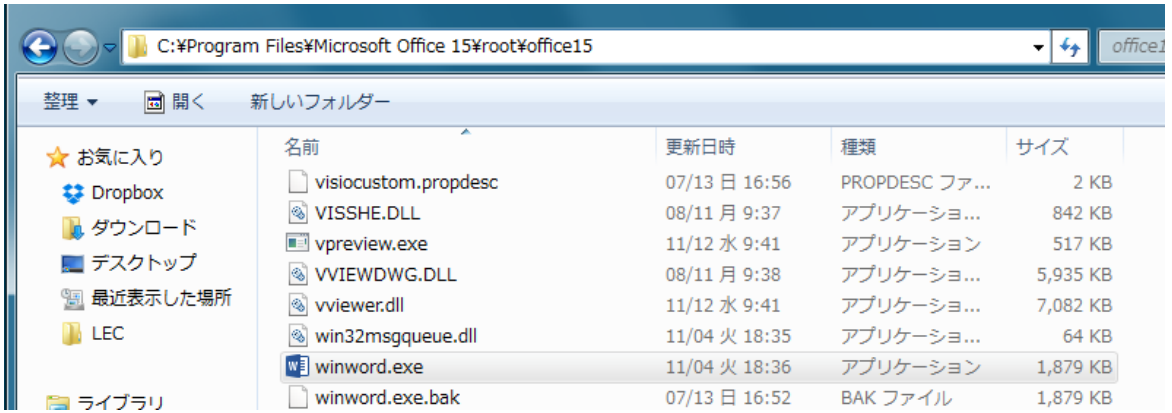
フォルダもしくはディレクトリ

- 下記はwindowsにおけるユーザーkaiyaの標準的なダウンロードファイルが置かれるフォルダ.
- 多くのOSでは，このような階層構造を用いてファイルを整理して保存している.



ほとんど全てのデータはファイル

- 広く使われるOS(Windows等)では，ほとんど全てのデータはファイルという単位で保管されている．
- 例えば，
  - MSワードの文書
  - MP3の音楽
  - MP4の動画
  - Cのプログラム
  - 実行可能プログラム (cc.exe やMSワード自体)
  - HTMLファイル

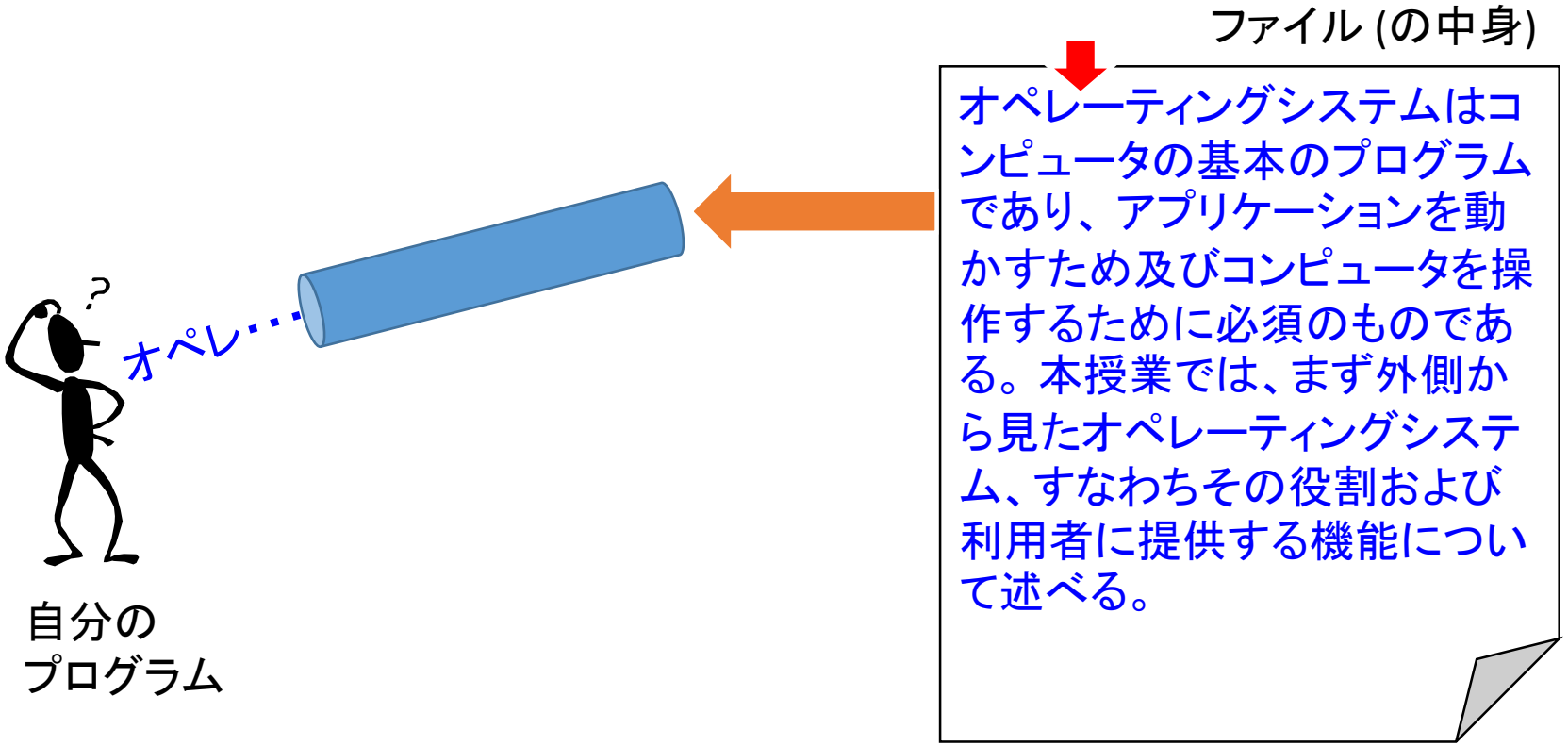


例えば，マイクロソフトワードは上記のファイルである．

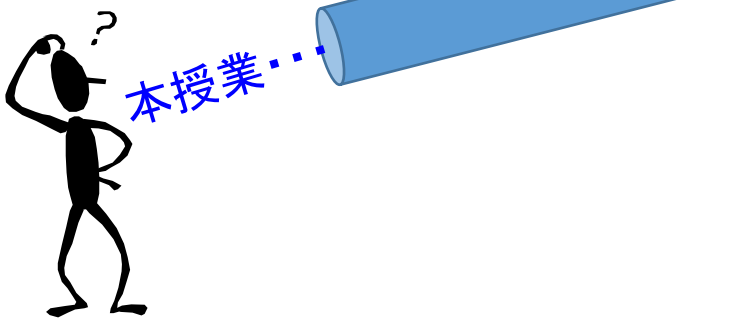
プログラムから見たファイル

- 自作のプログラムからは，ファイルは単なるデータの配列に見える．
- 配列のサイズはデータのバイト数に相当する．
- 配列なので，添え字に相当する値を用いて，ファイル内の任意の位置のデータを取り出すこともできる．
  - ランダムアクセス(Random Access)と呼ばれる．
- しかし，ファイルの先頭から順番にデータを取り出し，プログラム中でデータを利用するのが最も一般的な方法である．
  - 順次アクセス(Sequential Access)もしくはストーリム(Stream)と呼ばれる方法．
  - キーボード入力や画面出力と同様に，ファイルを取り扱おうとする考え方に由来する．

ストリームのイメージ 1/3



ストリームのイメージ 2/3

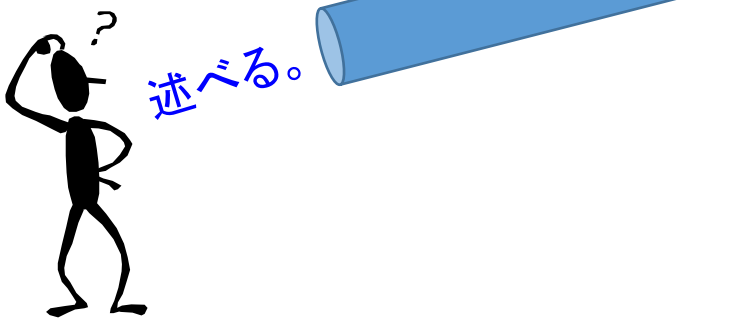


自分の
プログラム

ファイル (の中身)

オペレーティングシステムはコンピュータの基本のプログラムであり、アプリケーションを動かすため及びコンピュータを操作するために必須のものである。本授業では、まず外側から見たオペレーティングシステム、すなわちその役割および利用者に提供する機能について述べる。

ストリームのイメージ 3/3

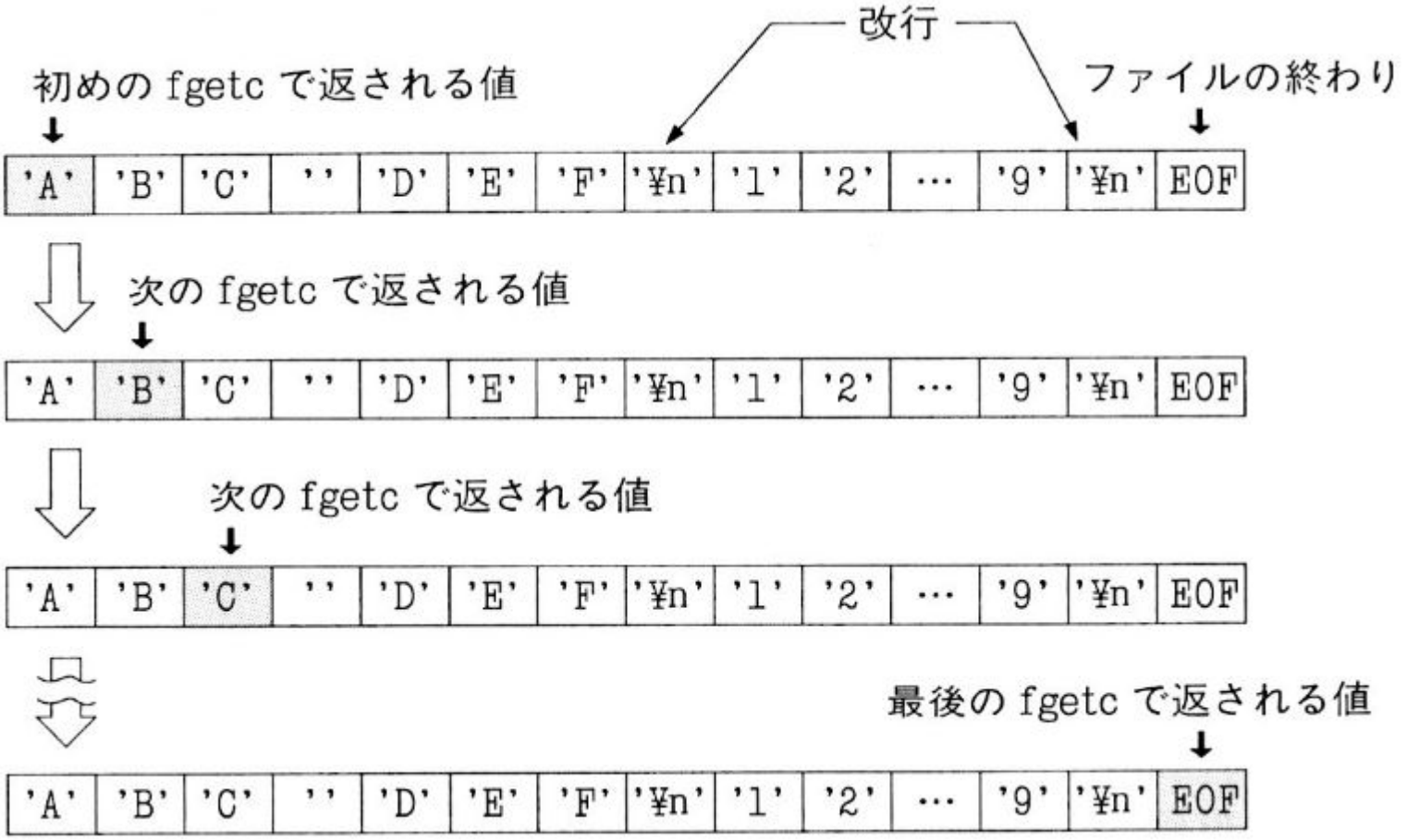


自分の
プログラム

ファイル (の中身)

オペレーティングシステムはコンピュータの基本のプログラムであり、アプリケーションを動かすため及びコンピュータを操作するために必須のものである。本授業では、まず外側から見たオペレーティングシステム、すなわちその役割および利用者に提供する機能について述べる。

Fig 12-3


改行
初めの fgetc で返される値
ファイルの終わり
'A' 'B' 'C' '' 'D' 'E' 'F' '¥n' '1' '2' … '9' '¥n' EOF
次の fgetc で返される値
'A' 'B' 'C' '' 'D' 'E' 'F' '¥n' '1' '2' … '9' '¥n' EOF
次の fgetc で返される値
'A' 'B' 'C' '' 'D' 'E' 'F' '¥n' '1' '2' … '9' '¥n' EOF
最後の fgetc で返される値
'A' 'B' 'C' '' 'D' 'E' 'F' '¥n' '1' '2' … '9' '¥n' EOF

ファイルを読むための関数群

- fopen
  - 前図で円柱で示したデータが通る道を作る関数.
  - データが通る道 = ストリーム
  - 読み，書き，新規ファイル作成共通で使われる.
- fgetc, fgets, fscanf
  - ストリームからデータを読み出す関数群.
  - それぞれ，一文字，一行，書式にあった形式を読み出す.
  - 読み込めると，読み込み位置(前図での赤い矢印)は読んだ分だけ先に進む.
- fclose
  - データが通る道(ストリーム)を撤去する関数.
  - プログラムからファイルが必要なくなったら必ず撤去しよう.

fopenの詳細

- FILE *fopen(const char *path, const char *mode);
- 返り値
  - データを取得するための道(前図の円柱)の端へのポインタが帰ってくる.
  - 失敗するとNULLが返る.

- 引数 path
  - ファイル名を指定.
  - 必要ならフォルダ名を含めたファイル名
    - 左上の例なら C:¥Users¥kaiya¥Downloads¥t2.pdf を指定.
- 引数 mode
  - 読み込みのみ"r"か，読み書き両方"rw"か等を指定.

fgetc, fgets, fscanf

- int fgetc(FILE *stream);
  - 一文字毎にデータをとってくる.
- char *fgets(char *s, int size, FILE *stream);
  - 基本，sで指定された文字配列にストリームから一行分の文字列を読み出す.
  - 改行文字もsに読み込む.
  - もし，一行がsizeを超えた場合，sizeまでしか読まない.
- int fscanf(FILE *stream, const char *format, ...);
  - scanfのストリーム版.

サンプル List 12-1

```
// 抜粋 list12-1.c
// ファイルの中身を画面に表示
char filename[FILENAME_MAX];
FILE *fp;
int c;

  printf("Filename? ");
  scanf("%s", filename);
  if((fp=fopen(filename, "r"))==NULL){
    printf("No such a file: %s¥n", filename);
    return -1;
  }

  while((c=fgetc(fp))!=EOF){
    putchar(c);
  }
  fclose(fp);
```

ファイルを書き込むの関数群

- fopen
  - 前図で円柱で示したデータが通る道を作る関数.
  - 読み，書き，新規ファイル作成共通で使われる.
  - モードは "w" で開く.
- fputc, fputs, fprintf
  - ファイルにデータを書き込む関数群.
  - それぞれ，一文字，一行，書式にあった形式を読み出す.
  - 書き込めると，書き込み位置(前図での赤い矢印)は書いた分だけ先に進む.
- fclose
  - データが通る道を撤去する関数.
  - プログラムからファイルが必要なくなったら必ず撤去しよう.

fputc, fputs, fprintf

- int fputc(int c, FILE *stream);
  - 一文字書き込み
- int fputs(const char *s, FILE *stream);
  - 一行，書き込み
  - sに改行文字が含まれれば，それを書き込む.
- int fprintf(FILE *stream, const char *format, ...);
  - 使い方はprintfに同じ.
  - 書式が決まったデータを書き出すのに便利.

書き込み例 List 12-2

```
// 抜粋 list12-2.c ファイルのコピー
char filename[FILENAME_MAX];
FILE *fp1, *fp2;
int c;

  printf("Input Filename? ");
  scanf("%s", filename);
  if((fp1=fopen(filename, "r"))==NULL){
    printf("No such a file: %s¥n", filename);
    return 1;
  }

  printf("Output Filename? ");
  scanf("%s", filename);
  if((fp2=fopen(filename, "w"))==NULL){
    printf("Cannot create the file: %s¥n", filename);
    return 1;
  }

  while((c=fgetc(fp1))!=EOF){
    fputc(c, fp2);
  }
  fclose(fp1);
  fclose(fp2);
```

ファイル削除，名前変更

- int remove(const char *pathname);
  - ANSIでの一般的なファイル削除関数
- int unlink(const char *pathname);
  - POSIX準拠のOS(ほとんどのOS)でremoveに同じ.
- int rename(const char *oldpath, const char *newpath);
  - 名前の変更.

- それぞれ成功すると返り値 0 か返る.

代入文の値利用

とても誤りが発生しやすい

代入文自体の値

- 変数には値が記録されている.
- 記録されている値は参照して利用できる.
  - たとえば printf 文 `printf("%d¥n", d)`
  - たとえば 代入文の右辺値 `x=y+1`
- 実は代入文自体にも値があり，参照できる．値は代入後の値.
  - 例 `x=(y=y+1)*3;`

  上記の意味は,
  - yの値を一つ増やす
  - 一つ増えたyの値を三倍したものをxとする.
- x++ と ++x の違い
  - 両方とも x=x+1 の略.

  ただし，代入文の値として,
  - x++ 一つ増やす前のxの値
  - ++x 一つ増やした後の値

  となる.

例題群

```
// valassert1.c
#include <stdio.h>

int main(void){
  int x, y;
  x=3;
  y=5;
  printf("%d %d¥n", x, y);
  // 以下の式で二個の代入を実行
  x=(y=y+1)*3;
  printf("%d %d¥n", x, y);
  return 0;
}
```

```
sh-3.1$ ./a.exe
3 5
18 6
sh-3.1$
```

```
// valassert2.c
#include <stdio.h>

int main(void){
  int x, y, z;
  x=3; y=5; z=3;
  printf("%d %d %d¥n", x, y, z);
  // 以下の式で二個の代入を実行
  x=(y++)*3; // ++する前の値を利用
  printf("%d %d %d¥n", x, y, z);
  z=(++y)*3; // ++した後の値を利用
  printf("%d %d %d¥n", x, y, z);
  return 0;
}
```

```
sh-3.1$ ./a.exe
3 5 3
15 6 3
15 7 21
sh-3.1$
```

代入文自体の値利用はお勧めしない

- 前頁の例のように，代入文自体の値を使うと，式がかなり複雑になる.
- これは，プログラム誤りの根源となるので，相当，慣れていない限り利用しないほうがよい.

参考: 代入文の値を使ったwhile

```c
#include <stdio.h>

int main(void){
  int i, n=10;
  i=0;
  while((i++)<n){
    // 条件判断ではiだが,
    // ブロック内では+1され済
    printf("%d¥n", i);
  }
  return 0;
} // nwhile1.c
```

```c
#include <stdio.h>

int main(void){
  int i, n=10;
  i=0;
  while(i<n){
    i++;
    printf("%d¥n", i);
}
  return 0;
} // nwhile2.c
```

```
sh-3.1$ ./a.exe
1
2
3
4
5
6
7
8
9
10
sh-3.1$
```

List 7-2

```
// List 7-2 p. 155
// コレも単に代入文の値を使っている
// 結果として c=getchar()を二回書かなくてよくなった
#include <stdio.h>

int main(void){
  int c;
  while((c=getchar())!='.'){
    printf("'%c'¥n", c);
  }
  return 0;
} // l7-2.c
```

```
// 参考 List 7-1
#include <stdio.h>
int main(void){
  int c;
  c=getchar();
  while( c!='.' ){
    printf("'%c'¥n", c);
    c=getchar();
  }
  return 0;
} // l7-1.c
```

switch文

switch文の必然性

- if文はかなり柔軟に条件を設定し，実行する命令を取捨選択できる.
- しかし，単純にいくつかの選択肢から一つ選択する等をif文で書くと若干煩雑になる.
  - 例えば変数に入っている曜日(を表す値)によって，実行命令を変更する等. (曜日の選択)
- switch文はこのような選択肢からの選択を直感的に記述することができる.

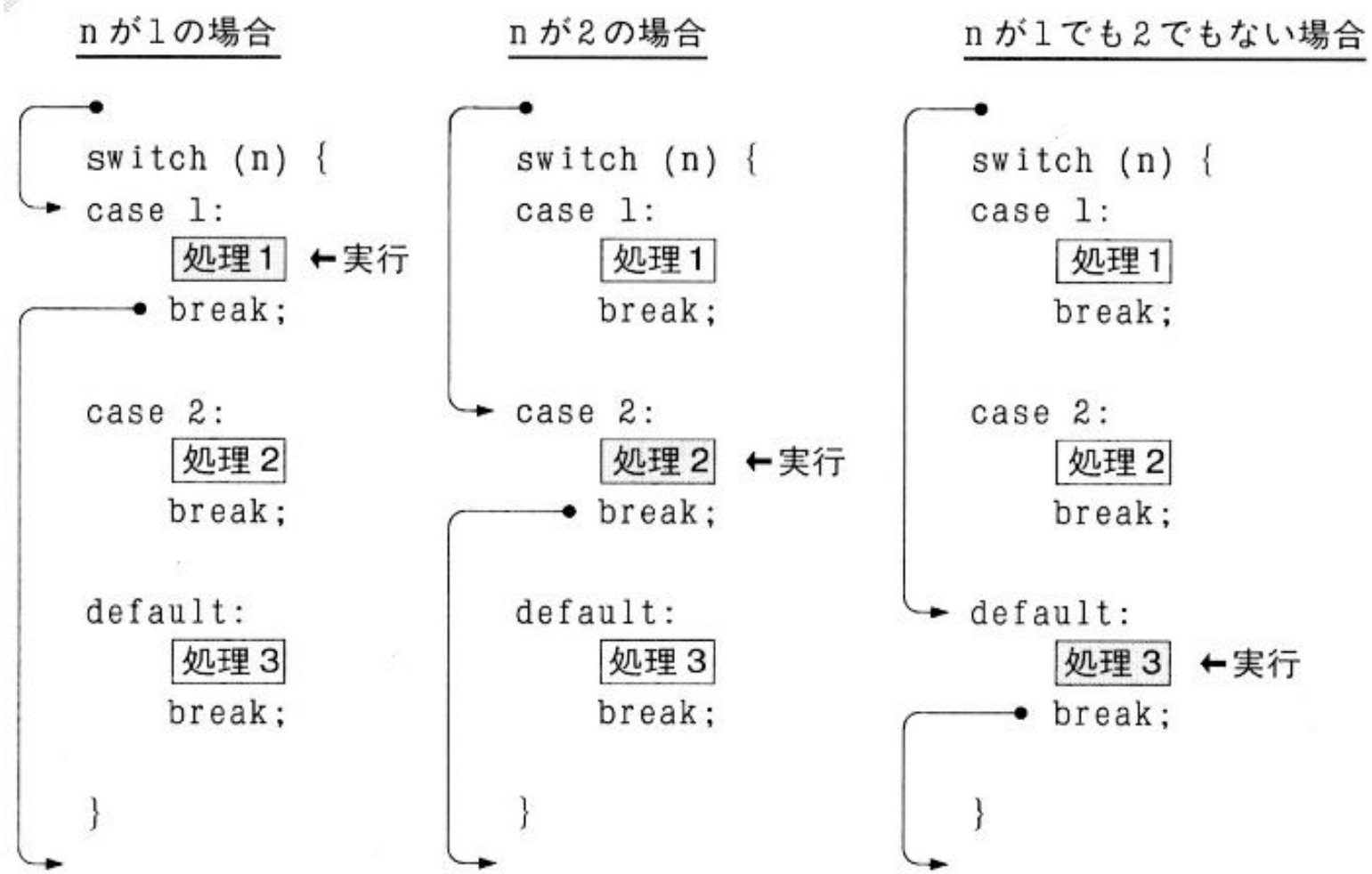
一般形と例

```
switch (式) {
case 定数1:
  命令文群1;
  break;
case 定数2:
  命令文群2;
  break;
case 定数3:
  // 以下，同様

default:
  命令群;
  break;
}
```

```
scanf("%d", &v);

switch (v) { // 値vについて
case 1: // vが1だったら
  printf("January¥n"); // 一月
  break;
case 2: // vが2だったら
  printf("February¥n"); // 二月
  break;
default: // vが上記以外だったら
  printf("March or later¥n");
  break;
}
// switchmonth.c
```

「省略された場合の選択」を意味します．
(債務不履行ではありません)

switch文の命令実行の流れ



教科書 p.109より

switch文の制限

- switch対象の式には，文字列を含む配列や構造体の場合分けは行なえない.
- 場合(case)には定数しかつかえない．変数はだめ.

- 個人的には，if文を使うほうをお勧めしたい.
- または，構造体等を使い，値と処理の対応表や，連想配列を作り，ループで検索するほうをお勧めしたい.
  - ただし，関数へのポインタ等が高度な文法が必要になる場合もある.

if文を使った同じ意味のプログラム

```
// switchif.c switchmonth.c と意味は同じ
#include <stdio.h>

int main(void){
int v;
scanf("%d", &v);

  if(v==1){ // vが1だったら
    printf("January¥n"); // 一月
  }else if(v==2){ // vが2だったら
    printf("February¥n"); // 二月
  }else{
    printf("March or later¥n");
  }
return 0;
}
```

条件演算子

- if (条件) 処理1 else 処理2 が単純にかける.

  条件? 処理1: 処理2

- 単に単純に書けるたけでなく，この書式全体が式の一部に使える.
  - 単純な条件判断の関数を名前をつけず定義できる感じ.
  - 普通の if else 文は，文であって式でないので，式の一部には使えない.

- 見た目はわかりにくいが，たまに超便利なことがある.
- 初心者にはあまりお勧めできないかも.

if文同様の使い方

```
int main(void){ // ifstate.c 普通の書き方
int v;
        scanf("%d", &v);
        if(v%2==0){ // 偶数 even number
                printf("%d is even. ¥n", v);
        }else{ // 奇数 odd number
                printf("%d is odd. ¥n", v);
        }
}
```

```
int main(void){ // ifexp.c 条件? 処理1: 処理2 の書き方
int v;
        scanf("%d", &v);
        v%2==0? // 偶数 even number
                printf("%d is even. ¥n", v):
                printf("%d is odd. ¥n", v);
} // あまりこの使い方はしない.
```

?:全体を式として利用

```
// ifexploop.c 式(expression)のif文
// 条件? 式: 式 を式として使う.
#include <stdio.h>

int main(void){
int i;
  for(i=0; i<10; i++){
    printf("%d is %s. ¥n", i, i%2==0? "even": "odd");
  }
}
```

これ全体が式として計算され，
結果が %s に入る.

```
bash-3.1$ ./a
0 is even.
1 is odd.
2 is even.
3 is odd.
4 is even.
5 is odd.
6 is even.
7 is odd.
8 is even.
9 is odd.
bash-3.1$
```

```
// if文でかくとかなりめんどい.
for(i=0; i<10; i++){
    printf("%d is ", i);
    if(i%2==0){
      printf("even. ¥n");
    }else{
      printf("odd. ¥n");
    }
  }
```

本日は以上

演習はありません