プログラミング通論'19 ＃10 – データの読み書きと整列

久野 靖 (電気通信大学)

2019.4.20

今回は次のことが目標となります。

- CSV ファイルの読み書きと取り扱いを経験する
- 基本的な (平易な) 整列アルゴリズムについて理解する

データの読み書き
CSV ファイルの形式

ここまで C 言語で様々なアルゴリズムと取り扱って来ましたが、そのアルゴリズムで取り扱うデータは「ちょっとしたテスト用のデータ」程度でした。しかし実際にはもちろん、様々なところから持って来た実験データや調査データをプログラムで取り扱いたいわけです。そのためには、ファイルからデータを読み込んだり、プログラムで処理した結果を書き出したりする必要があります。

データファイルの形式にも様々なものがありますが、ここでは一般に普及している (表計算ソフトで使われるという意味)、CSV(comma separated value) 形式を読み書きするという題材を扱います。表計算ソフトではデータを縦横のマトリクスに配置して扱いますが、CSV 形式ではそのデータを 1 行ずつ、各セルを「,」で区切って並べた形で表します。

CSV ファイルの形式 (2)

たとえば次の例は、日本のいくつかの都市の平均気温と年間降水量のデータを CSV 形式で表しています。

```
city,temprature,precitipation
Sapporo,8.2,1129.6
Sendai,11.9,1204.5
Tokyo,15.6,1405.3
Kanazawa,14.1,2592.6
Oosaka,16.3,1318.0
Hiroshima,16.2,1511.8
Kouchi,16.4,2582.4
Fukuoka,16.2,1604.3
Naha,22.4,2036.8
```

CSV ファイルの形式 (3)

ぎっちり詰まっていて見づらいですが、人間が見るものではないのでこれで別に良い訳です。そして、文字列と数値はとくに区別していません。

テキストの例はすべて **ASCII** の文字だけですが、日本語を扱う場合は **euc-jp** または **UTF8** ならバイト単位で見ていっても「,」の文字コードは「,」としてのみ使われるのでそのままで **OK** です (文字列の中身を処理するときはまた別の問題ですが、今回はバイト列のままでしか扱いません。

もう 1 つ、セルの中に「,」が出て来る場合はセルの文字列全体を「"...」で囲むことで区切りとして扱わせないようになっているのですが、ここでは簡単のためその機能は実装しません。[1]

[1] 演習問題としています。教員が面倒だと思うことは演習問題にしておくというのはこの業界の伝統ですので。

CSV を扱うデータ構造

CSV データをプログラムに読み込んだとして、それはどのようなデータ構造になっているのがいいでしょうか。ここでは行の並びは普通に配列にするとして、1 行の中は図 1 左のように、構造体の先頭にセルの個数 num があり、その先に文字列ポインタの配列 cell がくっついた形としました。

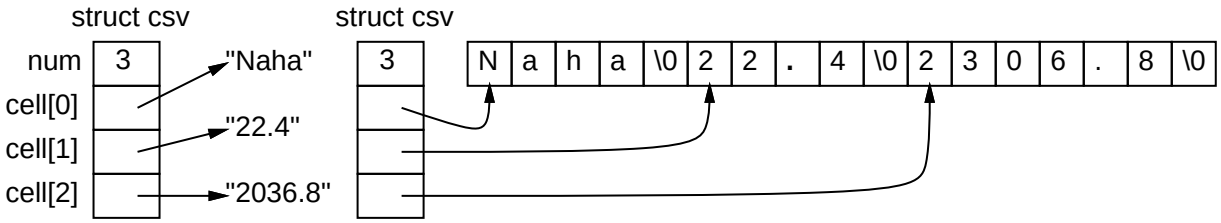

図 1: CSV の 1 行を表すデータ構造

実際のメモリ上では図 1 右のように、読み込んだ行がそのまま文字配列となっていて途中の「,」は各セルの文字列終わりを表すナル文字に変更し、各セルの先頭位置を cell[i] で指しています。

CSV を扱うデータ構造 (2)

この構造を使って CSV を読み書きする API のヘッダファイルを見ましょう。これまでと違って、ヘッダファイル中に struct csv のフィールドが書かれています。これは、読み込んだ CSV を自分でも扱いたいので、そのためにはフィールド num や cell を参照する必要があるからです。言い替えれば、中身を見られるようにしているため、これは抽象データ型ではない普通のデータ構造です。

```
// csv.h --- csv file read/write API.
struct csv { int num; char *cell[1]; };
typedef struct csv *csvp;
int csv_read(char *fname, int limit, csvp arr[]);
void csv_write(char *fname, int size, csvp arr[]);
```

CSV を扱うデータ構造 (3)

しかし、配列 cell の要素数が 1 になっていますが…これは、読み込む CSV に応じて (さらにもしかしたら行ごとに) セルの数は変わってくるので、いくつと書けないからです。しかし実際の領域そのものは、malloc で割り当てる時に指定すればいいので、いくつにでもできます。C 言語では配列の添字の範囲検査をしない (言語仕様上できない) ため、このような (他のフィールドと配列を直接くっつけた) 設計が可能です。[2]

関数の方ですが、csv_read はファイル名と struct csv のポインタの配列を渡し、そこに読み込んだ各行の構造体へのポインタを格納して行数を返します (ファイルが読めない、上限 limit で足りないなどのエラーがあった場合は負の数を返します)。csv_write は同じくファイル名と struct csv のポインタの配列と行数を渡し、ファイルに CSV 形式で内容を書き出します。

これまでの抽象データ型では使う側には内部構造が分からないので領域解放の関数も用意しましたが、にこのライブラリは内部を公開しているので、解放は各プログラムに任せることとしました。今回の例題ではファイルを 1 つ扱ったらもう終わるので解放はしていません。

[2] Java など範囲検査をする言語では、レコードの領域と配列の領域は別にする必要があります。

CSV ライブラリの実装

では実装を見てみましょう。取り込むヘッダの中に ctype.h がありますが、これはだいぶ後の方で使いますので当面保留してください。関数 readline と read1 はこのファイル内だけの下請けなので static と指定しています。いずれも FILE* を受け取り、そこから 1 行ぶんのデータを読み取ります。

readline は単に 1 行ぶんの文字列を配列 buf に読み込み、そのデータをずっと残しておくため、必要な領域を malloc で割り当ててそこに文字列をコピーして割り当てた領域の先頭を返します。なお、1 行読み込み関数 fgets は行の最後の改行を削除しないので、文字列の最後が改行だったらそれをナル文字に変更するという処理もしています。

CSVライブラリの実装 (2)

```
// csv.c --- csv file read/write API impl.
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "csv.h"
#define MAXLINE 1000
static char *readline(FILE *f) {
  char buf[MAXLINE], *str;
  if(fgets(buf, MAXLINE, f) == NULL) { return NULL; }
  int len = strlen(buf);
  if(buf[len-1] == '\n') { buf[--len] = '\0'; }
  str = (char*)malloc(len+1); strcpy(str, buf); return str;
}
static csvp read1(FILE *f) {
  char *arr[100], *s = readline(f); if(s == NULL) { return NU
  int i = 0;
  for(arr[i++] = s; *s != '\0'; ++s) {
    if(*s == ',') { *s = '\0'; arr[i++] = s+1; }
  }
  csvp r = (csvp)malloc(sizeof(struct csv) + i*sizeof(char*))
  r->num = i;
  for(i = 0; i < r->num; ++i) { r->cell[i] = arr[i]; }
  return r;
}
```

CSV ライブラリの実装 (3)

read1 は readline で 1 行を読み、各セル文字列の先頭を配列 arr に入れていきます。先頭は文字列の先頭で、あとは「’,’」がある毎にそこはナル文字に変更し、次の文字のアドレスを次のセルの先頭とします。行の最後まで終わったらこれでセルの数が分かりますから、malloc で構造体の領域を割り当てます。先に説明したように、セルの個数に応じて割り当てる領域を増やしています。[3]

read_csv はファイルを fopen で開き、read1 で繰り返し行を読み[4]配列に構造体ポインタを格納し、終わったらファイルを閉じてサイズを返します。なお、エラーがあった場合は適宜負の値を返しますが、ファイルが開けなかったときはすぐ-1 を返してよいですが、配列満杯のときはファイルを fclose て後始末する必要があるため、size に-2 を入れてループを抜けるようにしています。

[3]厳密には構造体の宣言で cell[1] として 1 個ぶんを確保してあるので、増やす量は (i-1)*sizeof(char*) でよいですが、なんとなく読みやすさのために 1 引くのは省略しました。大した無駄ではないので。

[4]「(line = read1(f))」は read1 から返された構造体のポインタを変数 line に入れますが、その入れた値を式の値として返します。もしそれが NULL だったらファイルの終わりなわけです。

CSVライブラリの実装 (4)

```
int csv_read(char *fname, int limit, csvp arr[]) {
  int size = 0; csvp line;
  FILE *f = fopen(fname, "rb"); if(f == NULL) { return -1; }
  while((line = read1(f)) != NULL) {
    if(size+1 >= limit) { size = -2; break; }
    arr[size++] = line;
  }
  fclose(f); return size;
}
void csv_write(char *fname, int size, csvp arr[]) {
  FILE *f = fopen(fname, "wb"); if(f == NULL) { return; }
  for(int i = 0; i < size; ++i) {
    fprintf(f, "%s", arr[i]->cell[0]);
    for(int j = 1; j < arr[i]->num; ++j) { fprintf(f, ",%s",
    fprintf(f, "\n");
  }
  fclose(f);
}
```

csv_writeは簡単で、ファイルを開いたあと各行ごとに、先頭のセルはそのまま、以後のセルは「,」に続いて出力し、行末の改行文字を出力するようになっています。実のところ、データを書き換えたりセルを増やすことを考えると、これを呼ぶより自前で出力する方が便利なことが多いでしょう。

CSV を読み込んで見る

では実際に先の CSV ファイルを読み込んでそのまま打ち出す、という例題を作りました (**のコメント行については後述)。

```
// csvread.c --- demonstration for csv_read.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "csv.h"

int main(int argc, char *argv[]) {
  csvp data[1000], p;
  int size = csv_read(argv[1], 1000, data);
  if(size <= 0) { return 0; }
//qsort(data+1, size-1, sizeof(csvp), cmp1); // **
  p = data[0];
  printf("%11s %11s %11s\n", p->cell[0], p->cell[1], p->cell[
  for(int i = 1; i < size; ++i) {
    p = data[i];
    printf("%11s %11.3f %11.3f\n",
           p->cell[0], atof(p->cell[1]), atof(p->cell[2])/12.
  }
  return 0;
}
```

CSV を読み込んで見る (2)

CSV ファイルでは 1 行目は「各カラムが何を意味するか」を表す文字列 (ヘッダ) であるのが通例で先のもそうなっているので、1 行目は別扱いしています。揃えるためには printf の書式を使いますが、そのためには実数は数にする必要があるので、atof で文字列から変換しています。実行のようすは次の通り。

```
% gcc8 cvsread.c cvs.c
% ./a.out jcitytemp.csv
       city  temprature precitipation
     Sapporo       8.200    1129.600
      Sendai      11.900    1204.500
       Tokyo      15.600    1405.300
    Kanazawa      14.100    2592.600
      Oosaka      16.300    1318.000
   Hiroshima      16.200    1511.800
      Kouchi      16.400    2582.400
     Fukuoka      16.200    1604.300
        Naha      22.400    2036.800
```

CSV を読み込んで見る (3)

演習 1 上の例題をそのまま動かせ。動いたら次のプログラムを作れ。実際に動かして確認すること。

a. 年間降水量のかわりに月平均降水量を打ち出す。
b. 最も降水量の多い都市のデータだけを打ち出す。
c. 最も平均気温が高い都市と低い都市のデータを打ち出す。
d. eps ライブラリと組み合わせてこのデータを好きに視覚化する。
e. その他好きなデータ処理をして打ち出す。

データの整列

前節のような「並んだデータ」に対し、特定の基準で整列する、というのは大変よくある処理です。整列アルゴリズムはあとで詳しくやりますが、その前に C の標準ライブラリにある **qsort** を使って整列をしてみましょう。qsort の宣言 (**stdlib.h** にあります) は次のものです。

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*comp)(const void*, const void*));
```

ここで size_t は整数と思ってください。base は整列する配列の先頭番地、nmemb は整列する要素数、size は要素 **1** 個あたりのバイト数です。これは、図 **2** 上のように、複数のフィールドから成る構造体が並んだ配列を整列する場合は要素の大きさが分かっている必要があるためです。

データの整列 (2)

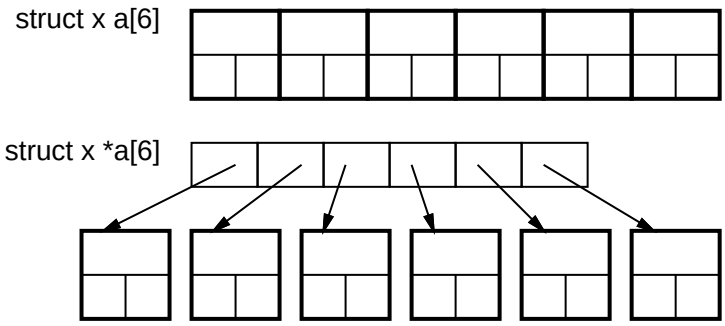

図 2: 構造体の配列と構造体ポインタの配列

ただ、整列は何回もデータを移動するため、構造体が直接並んだ配列だと構造体全体を何回もコピーする負荷が大きいです。そこで普通は、構造体は別の場所に置き (malloc で割り当てても別の構造体の配列に入っていてそのアドレスを取るのでもよい)、整列するのは「ポインタの配列」の方にすることが多いです (図 2 上)。前節までのデータ構造もそうなっていました。その場合、1 要素の大きさは sizeof(csvp) (ポインタ値は常に 8 バイト) になります。

データの整列 (3)

話題を戻して、最後の引数が難しそうですが、「間接参照すると、const void*型の引数2個を取り整数を返す関数になる」値 (関数へのポインタ値) を渡します。つまりqsortは要素を「昇順」に並べますが、どの基準で昇順かは、パラメタで渡した比較関数 (**comparator function**) を呼び出して決めます。[5]

比較関数は2つの番地を受け取り、その番地に入っている2つの値どうしを比べ、「前者が小さい」なら負、「等しい」ならゼロ、「前者が大きい」なら正の整数を返すことになっています。ところで「番地を受け取る」に注意してください。これは図2上のように大きな構造を並べた配列であれば構造体をコピーするのでなく番地だけ渡した方が効率的だからそうなっていますが、図2下のようにもともとポインタの配列であっても、そのポインタが入っている配列上の番地を渡してくることになり、実際のポインタを取り出すには1段間接参照する必要があります。

[5]「**const** が初出ですが、これはこの関数は受け取ったポインタの指す領域を変更しませんよ、ということを表します。今日の C ではこの「変更する/しない」をきちんと書くようにライブラリのヘッダが作られていますが、ここでは記述が込み入るのでライブラリで必要とする最低限だけ扱います。

データの整列 (4)

では実際にやってみましょう。データを都市名の順に並べ替えるとして、次のような比較関数cmp1を作りました。これをmainの上に置き、またstrcmpを使っているのでヘッダファイル**string.h**も**include**します。

```
static int cmp1(const void *x, const void *y) { // comp-fn.
  csvp a = *(csvp*)x, b = *(csvp*)y;
  return strcmp(a->cell[0], b->cell[0]);
}
```

内容は見ての通りで、パラメタのポインタをcsvp*にキャストしてから間接参照し、変数aとbにポインタを取り出します。そのあと、**2**つの都市名をstrcmpで比較して小/等しい/大の場合にそれぞれ負/ゼロ/正の値を返します。mainの中での呼び出しは先の例題にコメントで書いてありました。

データの整列 (5)

なぜ data+1 から size-1 個かというと、先頭の行はヘッダ (各列の見出し文字列だけが入っている) からです。動かしてみると、確かに都市名の ABC 順になっています。なお、整列に使う欄 (鍵ないしキー) は 1 つとは限りません。たとえばこの例でも温度が同じ都市があります。そこで「まず温度順、温度が同じものは降水量順」のように 2 番目以降の整列鍵を指定するわけです。

```
% ./a.out jcitytemp.csv
       city  temprature precitipation
    Fukuoka      16.200     133.692
  Hiroshima      16.200     125.983
   Kanazawa      14.100     216.050
     Kouchi      16.400     215.200
       Naha      22.400     169.733
     Oosaka      16.300     109.833
    Sapporo       8.200      94.133
     Sendai      11.900     100.375
      Tokyo      15.600     117.108
```

データの整列 (6)

演習 2 整列する例題をそのまま動かせ。動いたら次のように変更し、動かして動作を確認せよ。

a. 都市名の降順 (例題と反対の順) に並べる。
b. 降水量の多い順に並べる。
c. 平均気温の高い順に並べる。ただし平均気温が同じ場合は都市名の ABC 順に並べる (データを適当に修正して確認してよい)。

データの整列 (7)

d. ヨーロッパの各都市の毎月の平均気温と年平均気温を記録したCSVを授業サイトに用意した。冒頭部分は次のようになっている。これを「国名の降順、国が同じ中では都市名の昇順」で並べてCSVとして出力する。

```
Nation,City,1,2,3,4,5,6,7,8,9,10,11,12,AVG
Austria,Vienna,0.3,1.5,5.7,10.7,15.7,18.7,20.8,20.2,15.
Belgium,Brussels,3.3,3.7,6.8,9.8,13.6,16.2,18.4,18.0,14
...
Finland,Helsinki,-3.9,-4.7,-1.3,3.9,10.2,14.6,17.8,16.3
Finland,Kuopio,-9.2,-9.2,-4.1,2.0,9.1,14.5,17.5,15.0,9.
Finland,Oulu,-9.6,-9.3,-4.8,1.4,7.8,13.5,16.5,14.1,8.9,
...
```

e. 好きなデータを整列した上で何らかのデータ処理か視覚化をおこなう。

基本的な整列アルゴリズム＋α
選択ソート (単純選択法)

ここからは様々な整列アルゴリズムを見るため、データの方は「整数の配列を整列」という最も簡単な (分かりやすい) 形を取りましょう。整列順は昇順とします。

最初は単純選択法 (選択ソート、selection sort) と呼ばれるアルゴリズムです。これは図 3 にあるように、「1〜n の範囲で最小の値を 1 番目に置く」「2〜n の範囲で最小の値を 2 番目に置く」「3〜n の範囲で最小の値を 3 番目に置く」と繰り返して行くことで整列を完成させます。この「1 番目に置く」等のとき、その場所にこれまであった値を無くすとまずいですが、幸いそこに置こうとする値がこれまであった場所は空くので、そちらに移します (要は交換するわけです)。

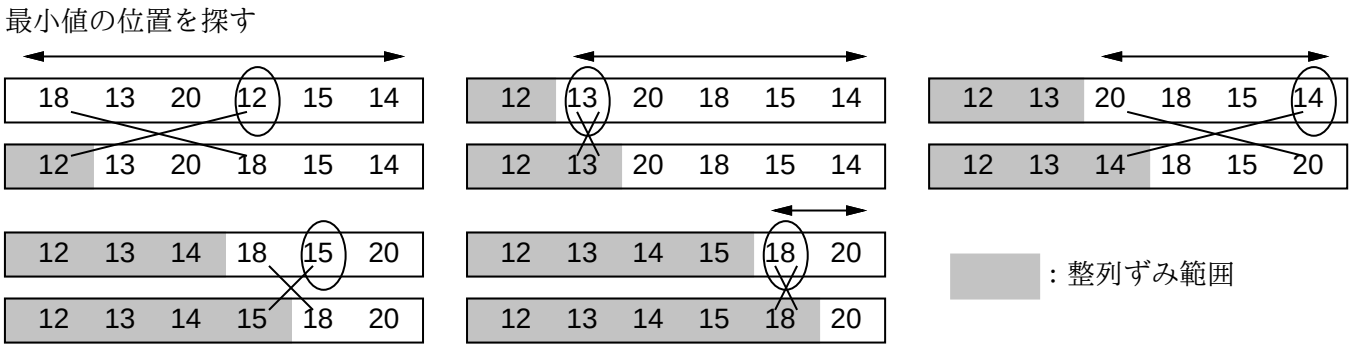

図 3: 単純選択法

選択ソート (単純選択法) (2)

ではコードを見てみましょう。配列 a の i 番と j 番を交換する関数 iswap がまず必要です。これはもう説明不要ですね。

次に、配列 a の i 番と j 番の範囲で「一番小さい値の位置」を調べる下請け関数 minrange を用意しました (これがある方が分かりやすい)。それには、a[i] をとりあえず min、i をとりあえず pos に入れ、それから i+1〜j の範囲について、もし a[k] が min より小さいならその値を min に入れ直し、同時に k を pos に入れ直します。

代入「=」は演算子であり、代入した値そのものを返すことから、「pos を入れ直しつつ a の添字としても使う」ように書いてありますが、このように短く書けるところは C 言語の発明です (今や大抵の言語でできますが)。調べ終わったら最後に位置 pos を返します。

選択ソート (単純選択法) (3)

```
static void iswap(int a[], int i, int j) {
  int x = a[i]; a[i] = a[j]; a[j] = x;
}
static int minrange(int a[], int i, int j) {
  int min = a[i], pos = i;
  for(int k = i+1; k <= j; ++k) {
    if(a[k] < min) { min = a[pos = k]; }
  }
  return pos;
}
void selectionsort(int n, int a[]) {
  for(int i = 0; i < n; ++i) { iswap(a, i, minrange(a, i, n-1
}
```

選択ソート (単純選択法) (4)

以上の準備ができたら本体は簡単で、「0〜n-1 の範囲の i について、i 番目に i〜n-1 番の範囲の最小を置く」だけです (もともとそういうアルゴリズムです)。このように、アルゴリズムとなるべく近い形にコードが書けると分かりやすいでしょう?

挿入ソート (単純挿入法)

次は単純挿入法 (挿入ソート、insertion sort) です。この方法は、1番目は1個だけで並んだ列だと考え、「2番目の値を取り上げ、1〜1番の列の正しい位置に挿入」「3番目の値を取り上げ、1〜2番の列の正しい位置に挿入」「4番目の値を取り上げ、1〜3番の列の正しい位置に挿入」と繰り返して行きます (図4)。

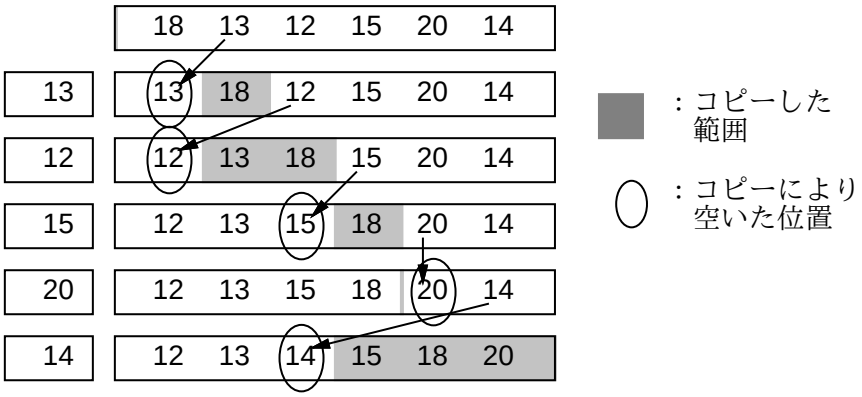

図 4: 単純挿入法

コードですが、「i番目の値を0〜i-1番 (整列済み) の正しい位置に移す」下請け shiftrange を用意しました。それには、i番目をxに取り出し、それからiを1つずつ減らしながら、i〜1番目にi-1〜0番の要素を入れて行きます (つまり1つずつ後ろにずらす)。

挿入ソート (単純挿入法) (2)

ただしずらすのは x より大きい要素だけで、そうでないものがでてきたらそこで止まります (x をそれより前に置いたら整列にならない)。最後まで全部ずらす場合もあります。最後に、ずらして空いた箇所に x を戻して終わりです。下請けができれば、本体は「1〜n-1 について、それを正しい位置に挿入」するだけです。

```
static int shiftrange(int a[], int i) {
  int x = a[i];
  for( ; i > 0 && a[i-1] > x; --i) { a[i] = a[i-1]; }
  a[i] = x;
}
void insertionsort(int n, int a[]) {
  for(int i = 1; i < n; ++i) { shiftrange(a, i); }
}
```

バブルソート

バブルソート (bubble sort) の原理は、昇順に並べるのですから、順に隣接する要素どうしを比較し、「大小が逆なら交換」することです。そうすると図5のように、1巡目が終わると最大値は右端に移動します。2巡目も同じようにすると、こんどは最大から2番目が最大の隣に移動します。

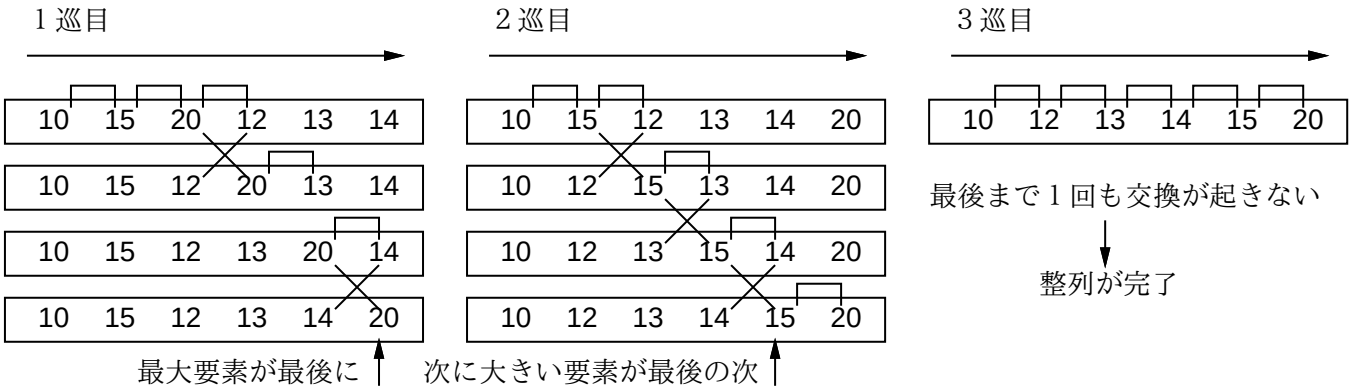

図 5: バブルソート

これを繰り返して行くとn-1巡すれば必ず終わりますが、運がよければそんなにやらなくても途中で完成することもあります。それは、1巡してみて「1回も交換が起きなかった」ことで分かります。

バブルソート (2)

このことを知るには「旗 (flag)」を使います。1 巡する前に旗を立て、比較した結果交換をするときは旗を降ろします。最後まで来て旗が降りていなかったら、1 回も交換していないので完了です。

次のコードでは、while 文を使うため、最初は旗の変数を「降りた状態」とし、while の条件は「旗が降りている間」で、while の中ではまず旗を立ててから一巡に入るようにしています。

```
void bubblesort(int n, int a[]) {
  bool done = false;
  while(!done) {
    done = true;
    for(int i = 1; i < n; ++i) {
      if(a[i-1] > a[i]) { iswap(a, i-1, i); done = false; }
    }
  }
}
```

コムソート

バブルソートは分かりやすいけれど手間が多く速くありません。それを少し変更しただけで速くできる例として、コムソート (**comb sort**) を見てみましょう。コム (**comb**) とは櫛のことで、「最初は髪がボサボサなので荒い櫛でとき、整ってきたら徐々に細かい櫛にする」というのが名前の由来だそうです。

どういうことかというと、先のバブルソートでは`a[i-1]`と`a[i]`を比較交換していましたが、これでは最初から目の細かい櫛で沢山ひっかかります。そこで距離`d`を導入し、`a[i-d]`と`a[i]`を比較交換することにして、`d`を最初は大きく (たとえば配列のサイズ)、一定比率で小さくしていき、最後は**1**になるようにします (図**6**)。**1**になったときにはバブルソートと同じなので、旗を用いて整列が完了するまで反復します。

コムソート (2)

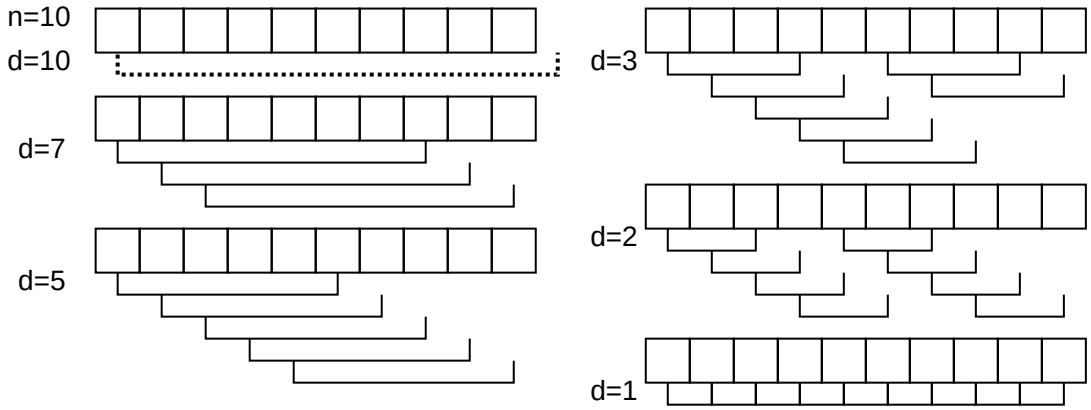

図 6: コムソート

ここで重要なのは「どれくらいの速さで」dを小さくしていくかです。あまり「一気に小さく」してしまうと、まだ髪が十分とけていないので並んでいない要素が多く、d=1で何回もとかすためバブルソートと変わりません。コムソートではdを「毎回$\frac{10}{13}$倍」にするのがよいとされています。

コムソート (3)

ということでコードを見てみましょう。バブルソートとほとんど変わりませんが、上述のようにd離れたところどうしを比較します。dの初期値はnとし、ループ内で1より大きい場合に毎回$\frac{10}{13}$倍します。そしてwhileの条件が「dが1より大きいか、または旗が降りている間」になっています。

```
void combsort(int n, int a[]) {
  int d = n; bool done = false;
  while(d > 1 || !done) {
    done = true;
    for(int i = d; i < n; ++i) {
      if(a[i-d] > a[i]) { iswap(a, i-d, i); done = false; }
    }
    if(d > 1) { d = d * 10 / 13; }
  }
}
```

単体テストと時間計測

では実際に整列ができているか調べるため、単体テストを作成しましょう。これまでと違い、大量データでテストするため、間違いがなければ**OK**、あれば大小が違っている最初の**5**個を出力するというふうにしました。さらに時間計測も行っています。整列を行う関数は関数ポインタで受け取り、それを時間計測しつつ呼び出しています。「`(*f)(n, a)`」は関数ポインタ`f`で指されている関数に引数`n`と`a`を呼び出すことを意味します。使用する配列は動的に割り当て、終了時に開放します。

`main`ではテストのため生成するデータ数をコマンド引数として受け取るようにしました。整列のコードはコピーして入れてもよいですが、プロトタイプ宣言を書いておいてコンパイル時に一緒にしてもよいです。

```
// test_sort.c --- unit test for sort algorithms.
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

(ここに各種ソートのコードかプロトタイプ宣言を入れる)
void expect_sort_iarray(int n, void (*f)(int m, int *a), char
  int c = 0, *a = (int*)malloc(n * sizeof(int));
  srand(time(NULL));
  for(int i = 0; i < n; ++i) { a[i] = rand()%10000; }
  struct timespec tm1, tm2;
  clock_gettime(CLOCK_REALTIME, &tm1);
  (*f)(n, a);
  clock_gettime(CLOCK_REALTIME, &tm2);
  for(int i = 1; i < n; ++i) {
```

```
    if(a[i-1] <= a[i]) { continue; } // correct order
    if(++c < 5) {
      printf("  wrong order at %d: %d > %d\n", i-1, a[i-1], a
    } else if(c == 5) {
      printf("  more wrong place omitted.\n");
    }
  }
  double dt = (tm2.tv_sec-tm1.tv_sec) + 1e-9*(tm2.tv_nsec-tm1
  printf("%s time=%.4f %s\n", c==0?"OK":"NG", dt, msg); free(
}
int main(int argc, char *argv[]) {
  int n = atoi(argv[1]);
  expect_sort_iarray(n, selectionsort, "selectionsort");
//expect_sort_iarray(n, insertionsort, "inserctionsort");
//expect_sort_iarray(n, bubblesort, "bubblesort");
//expect_sort_iarray(n, combsort, "combsort");
  return 0;
}
```

単体テストと時間計測 (2)

実行例を示します。整列はできているようです。

```
% ./a.out 1000
OK time=0.0015 selectionsort
OK time=0.0008 inserctionsort
OK time=0.0045 bubblesort
OK time=0.0002 combsort
```

単体テストと時間計測 (3)

これで要素数 **50000** にしたときの各種アルゴリズムの時間を手元のマシンで計測してみたものが表 **1** です。バブルソートは遅いですが、コムソートはとても速いことが分かります。

表 1: n = 50000 での各種整列の所要時間

アルゴリズム	時間 (秒)	時間計算量
selectionsort	3.770	$O(n^2)$
insertionsort	2.371	$O(n^2)$
bubblesort	14.824	$O(n^2)$
combsort	0.012	$O(n \log n)$

時間計算量ですが、選択ソート、挿入ソート、バブルソートはいずれも $O(n^2)$ になります。これは、外側ループの回数が n 回 (バブルソートは途中で終わるかも知れませんがいずれにせよ n に比例する回数)、内側ループの回数が前 **2** 者は $1 \sim n$ 回で平均 $\frac{n}{2}$ 回、バブルソートは常に n 回で、掛け算すると n^2 に比例する処理が必要になるためです。

単体テストと時間計測 (4)

コムソートはどうでしょう。d の値が n から初めて一定比率倍で減っていき最後は 1 になるので、1 になるまでの回数は $\log n$ に比例します。なので、「1 になったときには整列もほぼ終わっている」ようになっていれば (実際に $\frac{10}{13}$ 倍というのはそうなるように調整した値です)、この回数に 1 巡あたりの比較交換数 (徐々に多くなりますが最大 n です) を掛けて、$O(n \log n)$ になるわけです。

単体テストと時間計測 (5)

演習 3 コムソートの計算時間および時間計算量が、d に対して掛け算する値 (例では $\frac{10}{13}$) が変化するとどのように影響されるか、予想し、また実際に計測して検討しなさい。

演習 4 各アルゴリズムでランダムなデータを整列する時間については上で述べた通りであるが、これが「昇順に並んでいる」「降順に並んでいる」データだとそれぞれどのようになると思われるか。予想し、また実際に計測して検討しなさい。

演習 5 整列アルゴリズムについて好きなものを、先に出て来た qsort と同じ呼び方で使えるように変更し、qsrot の代わりに CSV ファイルの整列に使用して結果を確認しなさい。CSV を扱うプログラムは qsrot を渡すところ以外は変更しないこと。

本日の課題 10A

「演習1」～「演習2」で動かしたプログラム1つを含むレポートを本日中 (授業日の 23:59 まで) に提出してください。

1. sol または CED 環境で「/home3/staff/ka002689/prog19upload 10a ファイル名」で以下の内容を提出。
2. 学籍番号、氏名、ペアの学籍番号 (または「個人作業」)、提出日時。名前の行は先頭に「@@@」を付けることを勧める。
3. プログラムどれか1つのソースと「簡単な」説明。
4. レビュー課題。提出プログラムに対する他人 (ペア以外) からの簡単な (ただしプログラムの内容に関する) コメント。
5. 以下のアンケートの回答。

   Q1. CSV ファイルの読み込み方法が分かりましたか。

   Q2. 基本的な整列アルゴリズムを理解しましたか。

   Q3. リフレクション (今回の課題で分かったこと)・感想・要望をどうぞ。

次回までの課題10B

「演習1」～「演習5」(ただし10Aで提出したものは除外、以後も同様) の (小) 課題から選択して2つ以上課題をやり、レポートを提出しなさい。できるだけ複数の演習から選ぶこと。レポートは次回授業前日23:69を期限とします。

1. solまたはCED環境で「/home3/staff/ka002689/prog19upload10b ファイル名」で以下の内容を提出。
2. 学籍番号、氏名、ペアの学籍番号 (または「個人作業」)、提出日時。名前の行は先頭に「@@@」を付けることを勧める。
3. 1つ目の課題の再掲 (どの課題をやったか分かればよい)、プログラムのソースと「丁寧な」説明、および考察 (課題をやってみて分かったこと、分析、疑問点など)。
4. 2つ目の課題についても同様。
5. 以下のアンケートの回答。

   Q1. CSVファイルを読み込んでデータ処理できそうですか。

   Q2. プログラムのコードを見て時間計算量が分かりそうですか。

   Q3. リフレクション (今回の課題で分かったこと)・感想・要望をどうぞ。