プログラミングおよび実習II・第 5 回

2003 年 10 月 20 日

1 バブルソートの復習

1.1 実力チェック

次の関数 bubblesort は、前回の「1.1 実力チェック」での実現の方針に基づいて、与えられた配列の要素を昇順に並べ替えるものです。

効率化した bubblesort のプログラム

```
 1 void bubblesort(int data[], int num)
 2 {
 3     int last, pos, i;
 4     int tmp;
 5
 6     for (last = num-1; 0 < last; last = pos) {
 7         pos = 0;
 8         for (i = 0; i < last; i++) {
 9             if (data[i] > data[i+1]) {
10                 tmp = data[i];
11                 data[i] = data[i+1];
12                 data[i+1] = tmp;
13                 pos = i;
14             }
15         }
16     }
17 }
```

この関数定義に次の 2 点の変更を加えなさい。

(1) 整列対象の配列を int 型の配列ではなく、double 型の配列にする。

(2) 配列中の 2 つの要素の大きさの比較 (このプログラムでの 9 行目) を行った回数を int 型の返り値として (この関数が) 返すようにする。

ただし、解答用紙のプログラム名は「prog5-1.c」とし、関数 bubblesort の (変更後の) 定義のみを書きなさい。

2 クイックソートのプログラミング

効率がよく実用的な整列 (ソーティング) アルゴリズムの 1 つである**クイック (quick) ソート**と呼ばれる整列アルゴリズムのプログラミングをしてみましょう[1]。

2.1 クイックソートのアルゴリズム

配列 d 中にいくつかの整数データが格納されているとします。これの配列の添字 h から添字 t までの区間 ($d[h \ldots t]$) を、先頭側により小さいデータが来るように (昇順に) 並び替えたいとします。

[1] 教科書「C 言語によるアルゴリズムとデータ構造入門」の 116 ページからを適宜参照してください。一般に「クイックソート」と呼ばれるアルゴリズムには、細く見るとたくさんの種類があります。ここで紹介するのはその 1 例です。

実現の方針 (1) 与えられた区間中のある要素の値を基準値 r として採用します。

(2) この区間中の要素を適当に並び替えて、基準値 r より小さいか等しい要素だけからなる部分 $d[h \ldots i{-}1]$、基準値 r と等しい要素だけからなる部分 $d[i \ldots j]$、基準値 r より大きいか等しい要素だけからなる部分 $d[j{+}1 \ldots t]$ の 3 つの部分に分割する[2]。このためには、まず、与えられた区間の先頭から順に要素の値を調べて $d[i] \geqq r$ であるような要素 $d[i]$ を見つける。また、同様に区間の末尾から順に調べて $d[j] \leqq r$ であるような要素 $d[j]$ を見つける。見つかった i と j について、$i < j$ が成り立っていれば、$d[i]$ と $d[j]$ の値を交換し、i を 1 増やし、j は 1 減ずる。この操作を $i \geqq j$ となるまで続ければ、与えられた区間全体を先に述べたような 3 つの部分に分割することができる。基準値 r の選び方から、分割によってできた両側の 2 つの区間 $d[h \ldots i{-}1]$ と $d[j{+}1 \ldots t]$ の長さは、それぞれ元の区間よりも短くなる。

(3) この 2 つの区間それぞれに対して (1) と (2) の手順を再帰的に適用すれば、各区間の大きさは次第に小さくなっていき、ついにはその区間に含まれる要素はすべて同じ値を持つようになる。すべての区間をこのような状態にすることで、最初に与えられた区間全体を昇順に整列させることができる。

アルゴリズム (1) 与えられた整数配列を `d` とし、昇順に整列したい区間の先頭位置 (添字) を `h`、末尾位置 (添字) を `t` とする。これらを引数として、次の 6 つ手順を実行するサブルーチン (関数) を `quicksort` として定義する。

(1-1) 整数変数 `r`、`tmp`、`i`、`j` を用意する。

(1-2) もし、`h` $\geqq$ `t` であれば、整列の必要はないので、この `quicksort` の処理を呼び出した側へ戻る。

(1-3) 与えられた区間の中央の要素の値 `d[(h+t)/2]` を変数 `r` へ代入し、これを「実現の方針」での基準値とする。

(1-4) `i` に `h` を、`j` に `t` を代入する。

(1-5) 次の 5 つの手順を繰り返す。

(1-5-1) `d[i] < r` が成り立っている限り、`i` を 1 増やす。

(1-5-2) `r < d[j]` が成り立っている限り、`j` を 1 減ずる。

(1-5-3) `j <= i` が成り立っていれば、この繰り返しを終了する。

(1-5-4) `tmp` を使って、`d[i]` と `d[j]` の値を交換する。

(1-5-5) `i` を 1 増やし、`j` を 1 減ずる。

(1-6) `d`、`h`、`i`-1 の 3 つを実引数として、サブルーチン `quicksort` (つまり自分自身) を呼び出す。

(1-7) `d`、`j`$+1$、`t` の 3 つを実引数として、サブルーチン `quicksort` (つまり自分自身) を呼び出す。

[2] $j = i{-}1$ または $j = i$ となります。$j = i{-}1$ の場合は、区間 $d[j \ldots i]$ の長さは 0、$j = i$ の場合は 1 となります。

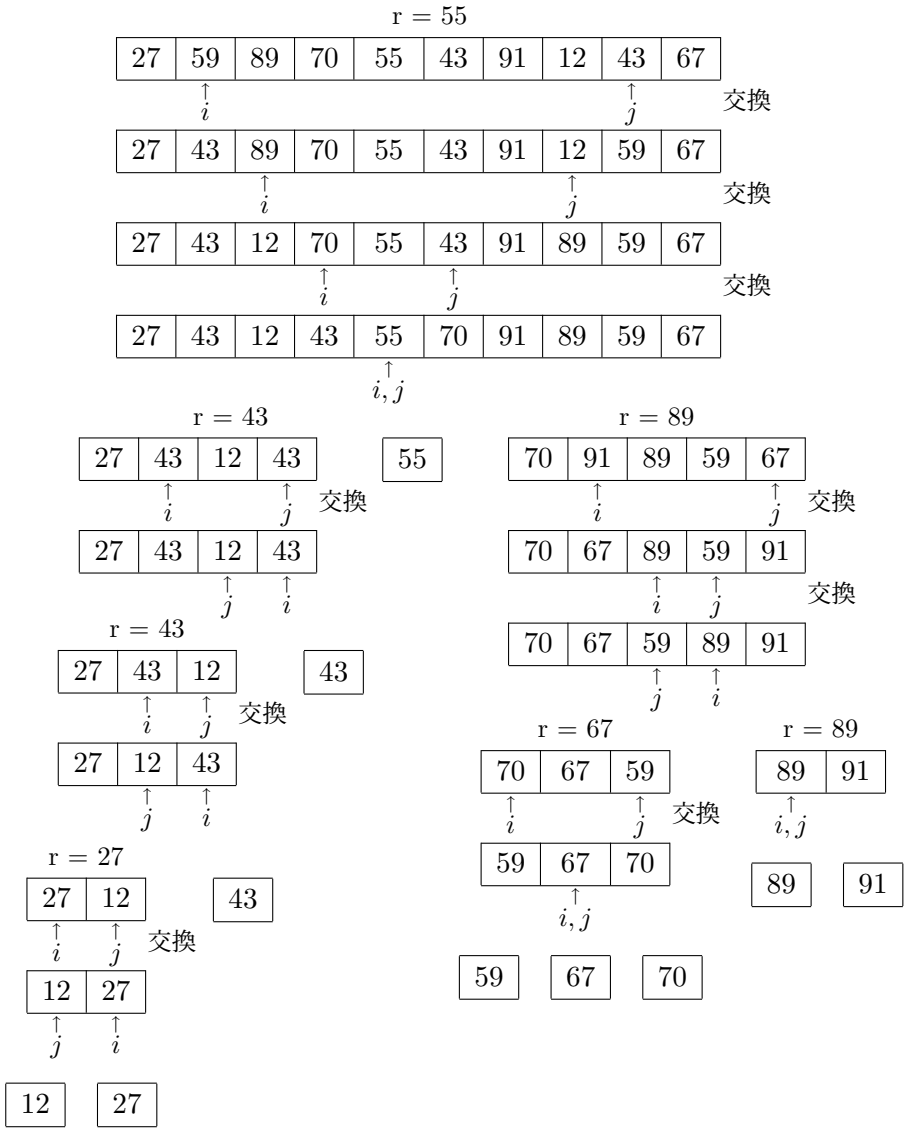

図 1: クイックソート法による整列が進む様子

図 1 は、与えられた配列 (あるいは区間) の中央の要素 (d[(h+t)/2]) の値を基準値 r として選ぶようなクイックソートを行った場合の整列過程である。

関数 getref と quicksort のプログラム例

```
 1 void quicksort(int d[], int h, int t)
 2 {
 3     int r, tmp;
 4     int i, j;
 5
 6     if (h >= t)
 7         return;
 8
 9     r = d[(h+t)/2];
10     i = h;
11     j = t;
12     while (1) {
13         while (d[i] < r)
14             i++;
```

```
15          while (r < d[j])
16              j--;
17          if (j <= i)
18              break;
19          tmp = d[i];
20          d[i++] = d[j];
21          d[j--] = tmp;
22      }
23
24      quicksort(d, h, i-1);
25      quicksort(d, j+1, t);
26  }
```

3 演習問題

以下の 2 つの演習問題に取り組みなさい。プログラムが完成したら、これまで通り mprog2 コマンドを実行して、自分が作成したプログラムを提出してください。

3.1 prog5-1.c

前回作成した prog4-1.c を prog5-1.c にコピーし、今回の「1.1 実力チェック」で行ったのと同様の変更をこのプログラム prog5-1.c に加えるとともに、関数 showdata や関数 main の定義も変更して、小数部付きの数値を入力すると、整列の過程と、その過程で行われた要素の値の比較の総数を次の実行例のように表示するようにしなさい。ただし、配列中の要素の値は小数点以下 2 桁まで表示するようにしなさい[3]。

prog5-1.c の実行例

```
s1542h017% ./prog5-1
 1番目のデータ => 27.2
 2番目のデータ => 18.35
 3番目のデータ => 43
 4番目のデータ => 7.77
 5番目のデータ => 67.02
 6番目のデータ => 72.39
 7番目のデータ => 89.99
 8番目のデータ => .
 27.20, 18.35, 43.00,  7.77, 67.02, 72.39, 89.99
 18.35, 27.20,  7.77, 43.00, 67.02, 72.39, 89.99
 18.35,  7.77, 27.20, 43.00, 67.02, 72.39, 89.99
  7.77, 18.35, 27.20, 43.00, 67.02, 72.39, 89.99
比較の総数 = 9
s1542h017%
```

注意: このプログラムは mprog2 コマンドで提出すると、教員か TA のチェックを受けるように要求されます。mprog2 コマンドが表示する指示に従ってください。

[3]たとえば、関数 printf を使って、double 型の値を 6 文字分の幅に小数点以下 2 桁まで表示するための出力書式文字列は "%6.2f" となります。

3.2 prog5-2.c

今回の「2. クイックソートのプログラミング」を参考にして、標準入力 (キーボード) から読み込んだ小数部付きの数値データを小さい順に並び替えて 1 行に表示する C プログラム prog5-2.c を作りなさい。ただし、double 型の配列 d、int 型の 2 つの値 h、t に対し、quicksort(d, h, t); のように呼び出すと、配列 d の添字 h から t までの区間をクイックソート法で昇順に整列するような関数 quicksort を定義し、この関数を main から呼び出すようなプログラムにしなさい。また、整列後のデータはそれぞれ小数点以下 2 桁まで表示するようにしなさい。

prog5-2.c の実行例

```
s1542h017% ./prog5-2
 1番目のデータ => 27.2
 2番目のデータ => 18.35
 3番目のデータ => 43
 4番目のデータ => 7.77
 5番目のデータ => 67.02
 6番目のデータ => 72.39
 7番目のデータ => 89.99
 8番目のデータ => .
  7.77, 18.35, 27.20, 43.00, 67.02, 72.39, 89.99
s1542h017%
```

余裕のある受講者は、prog5-2.c で定義した関数 quicksort を、前回の prog4-3.c と同様な方法で 10000 回テストするプログラム prog5-3.c を作成して、コンパイル、実行してみなさい。

prog5-3.c の実行例

```
s1542h017% ./prog5-3
すべてのチェックに合格しました。
s1542h017%
```

プログラミングおよび実習Ⅱ・第 5 回・終り

付録 A. C の関数

プログラムが行うべき全体の仕事の一部分を担うような一連の処理 (コンピュータへの指示) を行う働きを持つものを C では**関数**と呼びます。関数が行う (コンピュータにさせる) べき仕事を記述したものを、その関数の**関数定義**と呼びます。関数を定義しておくと、その関数を呼び出すことによって、その定義に記述されている仕事をさせたり、その計算結果を利用したりすることができます。これを**関数呼び出し**と言います。C では、いくつもの関数を定義することができ、ある関数の行うべき仕事の一部を、別の関数の呼び出しを使って実現することができます。

C では、よく必要となる仕事に関しては、それを行ってくれる関数の定義があらかじめ用意されています。これらの関数を**標準ライブラリ関数**と呼びます。文字列の入出力を行う scanf や printf、平方根の計算を行う sqrt などは、この標準ライブラリ関数の一部です。

式　プログラムの中で何らかの**値**を表すものを**式**と呼びます。式は、**定数**、**変数**、**関数呼び出し**、および、それらを四則演算などの**演算子**で組み合わせて構成されます。プログラム中の式を計算して、その計算結果を求めることを (式の) **評価**と呼びます。式が評価される過程では、この**副作用**として、変数の代入が行われたり、関数の呼び出しが起ったりすることもあります。

文　プログラム中でのコンピュータに対する指示の記述を**文**と呼びます。たとえば「変数 y の値に 1 加えたものを変数 x へ代入せよ」という指示を「`x = y + 1;`」という文として書くことができます。また「文字列 `"Hello!\n"` を引数として関数 `printf` を呼び出せ」という指示が「`printf("Hello!\n");`」となります。このような基本的な文を基に、場合分けの処理を行う `if` 文や `switch` 文[4]を構成したり、繰り返しの処理を行う `while` 文や `for` 文、`do` 文を構成することで、複雑な指示を記述することができます。また、文が実行される順序を制御する `break`文 や `continue` 文、`goto` 文、関数呼び出しを完結させる `return` 文などを利用することもできます。

C プログラムでは、演算子や括弧、「`;`」などの記号の両側の空白は (いくつ) あってもなくても構いません。また、1 つの行に複数の宣言や文を書いても構いませんし、複数の行にわたって 1 つの宣言や文を書いても構いません。

C プログラム全体の形　C のプログラムは**関数**や**変数**の**定義**の集まりです[5]。実行可能[6]な C プログラムの中には、`main` という名前の関数が定義されていなければなりません。もっとも単純な C プログラムは、この `main` という関数の定義のみからなる次のような形となります。

```
main()
{
    ⋮
}
```

関数定義　C の**関数定義**の基本形は次のようなものです。

返り値の型 関数名(仮引数$_1$ の宣言, 仮引数$_2$ の宣言, ..., 仮引数$_n$ の宣言)
{
　　関数定義本体 — この関数が呼ばれた時に実行される (宣言や) 文の並び
}

「返り値の型」は、この関数が (呼び出し元に) 返す値のデータ型を示します。「返り値の型」を省略すると、そこに `int` が書いてあるものとして扱われます。値を返すことのない関数の場合、「返り値の型」は `void` となります。関数名の後に（ と ）で囲まれ、「`,`」で区切られて並んでいる「**仮引数** (かりひきすう) の宣言」は、それぞれ変数の宣言のように働き、ここで宣言された仮引数は「関数定義本体」の中で通常の変数として使用することができます。仮引数は、次節で説明するように、この関数が呼び出される度に、その時の**実引数** (じつひきすう) で初期化されます。

---

[4] 2 年生前期までの科目ではまだ出てきていないかも知れません。

[5] プログラム中には、これ以外に、関数定義や変数定義を行うために必要な**型定義**、**関数宣言**、**変数宣言**、**前処理指令**などが現れます。

[6] もちろんコンパイルしてからの話ですが ...

引数のない関数を定義する場合、「関数名」に続く（ と ）の間には void を書いておきます。（ と ）の間に何も書かない場合は、この関数はどのような引数を取るのか指定しないことになります。

関数呼び出し　関数は式の一部として、

関数名(式$_1$，式$_2$，...，式$_n$)

ような形式で呼び出すことができます、このとき、 式$_1$、 式$_2$、 ...、 式$_n$ の計算結果は、この関数呼び出しにおける**実引数** (じつひきすう) と呼ばれ、関数定義の対応する仮引数 (変数) にそれぞれ代入されてから「関数定義本体」が実行されます。「関数定義本体」の中で、次の形の return **文**が実行されると、そのときの「式」の値が計算されて、これがこの関数の**返り値** (戻り値) となります。

retrun 式;

return 文が実行されると、プログラムの実行はこの関数を呼び出した側に復帰し、関数の返り値が関数呼び出し式「関数名(式$_1$，式$_2$，...，式$_n$)」の値となります。

値を返す (「返り値の型」が void ではない) 関数の定義の「関数定義本体」の実行は必ず「return 式;」が実行されて終らなければなりません。そうでないとこの関数の返り値は予測がつかないものとなります。値を返さない (「返り値の型」が void の) 関数の場合は、「式」のない return 文「return;」が実行されるか、「関数定義本体」の実行がすべて終ると、その関数を呼び出した側へ戻ります。

関数宣言　プログラム中にまだ定義が現れていない関数を利用したい場合は、次のような形式で、呼び出される関数の**関数宣言**をしておかなければなりません[7]。ただし「extern」や「仮引数の宣言」の中の引数名は省略することもできます。

extern 返り値の型 関数名(仮引数$_1$ の宣言，仮引数$_2$ の宣言，...，仮引数$_n$ の宣言);

標準ライブラリ関数については、そこで用意されている関数の関数宣言が**ヘッダファイル**と呼ばれるファイルにあらかじめ記述されていますので、このファイルを「#include」[8]でソースプログラム中に取り込むだけで関数宣言を済ますことができます。たとえば、標準入出力ライブラリ関数である scanf や printf を使うためには、プログラムの冒頭に

```
#include <stdio.h>
```

の 1 行を書いておきます。

付録 B. 前回の演習問題 prog4-2.c のプログラム例

prog4-2.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
```

[7] まだ定義も宣言もされていない関数に対する関数呼び出しを行うと、呼び出される関数は int 型の値を返り値として返すものと仮定されてコンパイルが行われます。このため、以降のプログラムで、その関数が別の型を返り値とするものと定義されたり宣言されたりすると、コンパイルエラーとなったり、コンパイラによって警告が出されたりします。

[8] # のように、(スペースやタブを無視して) # から始まる行はコンパイラの動作を制御するための指示で、**前処理指令**と呼ばれます。

```
 4 #define MAX_DATA        1000
 5
 6 void showdata (int data[], int num)
 7 {
 8     int i;
 9
10     if (num > 0) {
11         printf("%3d", data[0]);
12         for (i = 1; i < num; i++)
13             printf(",%3d", data[i]);
14     }
15     printf("\n");
16 }
17
18 void showlast (int last)
19 {
20     int i;
21
22     for (i = 0; i < last; i++)
23         printf("    ");
24     printf("  ^last\n");
25 }
26
27 void bubblesort(int data[], int num)
28 {
29     int last, pos, i;
30     int tmp;
31
32     for (last = num-1; 0 < last; last = pos) {
33         showdata(data, num);
34         showlast(last);
35         pos = 0;
36         for (i = 0; i < last; i++) {
37             if (data[i] > data[i+1]) {
38                 tmp = data[i];
39                 data[i] = data[i+1];
40                 data[i+1] = tmp;
41                 pos = i;
42             }
43         }
44     }
45 }
46
47 int main ()
48 {
49     int data[MAX_DATA];
50     int num;
51
52     for (num = 0; num < MAX_DATA; num++) {
53         printf("%2d番目のデータ => ", num+1);
54         if (scanf ("%d", &data[num]) != 1)
55             break;
56     }
57
58     bubblesort(data, num);
59     showdata(data, num);
60     return EXIT_SUCCESS;
61 }
```