プログラミングおよび実習II・第 6 回

2003 年 10 月 27 日

1 構造体の整列

前回までに見てきたソーティングプログラムはすべて、整数や実数などの単純なデータを要素とする配列を整列させるものでした。今回はもう少し複雑なデータをソーティングしてみましょう。

1.1 点数と名前の組からなるデータの列を点数の小さい順に整列する

標準入力 (キーボード) から、点数と名前を次々と読み込み、読み込みが終った時点で、それまでに読み込んだデータの組を点数の小さいものから順に出力するプログラム prog6-1.c を作ってみましょう。このプログラムの実行例は次のようなものになります。

prog6-1.c の実行例

```
s1542h017% ./prog6-1
 1番目の点数と名前 => 35 Hiroshi
 2番目の点数と名前 => 23 Koji
 3番目の点数と名前 => 67 Satoshi
 4番目の点数と名前 => 87 Junko
 5番目の点数と名前 => 64 Ichiro
 6番目の点数と名前 => 89 Mari
 7番目の点数と名前 => 73 Daisuke
 8番目の点数と名前 => .
 23 Koji
 35 Hiroshi
 64 Ichiro
 67 Satoshi
 73 Daisuke
 87 Junko
 89 Mari
s1542h017%
```

このプログラムは、読み込んだ点数と名前の組を次のような構造体型のデータとして記憶しています。

```
#define NAME_LEN        (100)

typedef struct {
    char name[NAME_LEN];
    int score;
} Record;
```

また、このプログラム prog6-1.c の関数 main の定義は次のようなものです。

```
#define MAX_DATA        (1000)

int main ()
{
    Record data[MAX_DATA];
    int num;

    for (num = 0; num < MAX_DATA; num++) {
        printf("%2d番目の点数と名前 => ", num+1);
        if (scanf("%d %99[^\n]", &data[num].score, data[num].name) != 2)
            break;
```

```
    }
    quicksort(data, 0, num-1);
    showdata(data, num);
    return EXIT_SUCCESS;
}
```

scanf の入力書式文字列にある「%99[^\n]」は、改行文字以外の文字 (の文字コード) を、対応する引数で指定されたアドレスへ最大 99 文字まで読み込むための指定です。このプログラム例の場合は、配列 data[num].name の先頭要素から順に、改行文字が読み取られる直前まで文字 (の文字コード) が最大 99 文字分格納されていき、最後に文字列の終りを示す 0 (文字コードとしての 0) が格納されます。一般に %[という変換指定子は、これと対応する] との間に書かれた文字が続く限り入力された文字列を読み取ります。[の直後が ^ の場合は、それ以降に書かれた文字以外の文字が続く限り読み取りを続けます。%[A-Za-z0-9] や [^0-9] のように - を使って、読み取りを続ける文字コードの範囲 (あるいは除外する文字コードの範囲) を指定することもできます。

関数 main から呼び出されている quicksort は、

```
void quicksort(Record d[], int h, int t);
```

と宣言することができる関数であり、 (Record 型の構造体を要素とする) 配列 d の、添字が h から t までの区間を、各要素の score の部分が昇順となるように (クイックソートのアルゴリズムで) 整列を行います。また、関数 showdata は、たとえば

```
void showdata (Record data[], int num)
{
    int i;

    for (i = 0; i < num; i++)
        printf("%3d %s\n", data[i].score, data[i].name);
}
```

のように定義されるもので、Record 型の配列 d の先頭から num 個数分の要素について、 1 行に 1 つずつ、各要素の点数 (score) と名前 (name) を (前ページの実行例のように) 表示します。

1.2 実力チェック

このプログラム prog6-1 に補うべき関数 quicksort の定義を書きなさい。ただし、解答用紙のプログラム名は「prog6-1.c」とし、この関数の定義のみを書きなさい。

2 ポインタを利用して効率よく要素を交換する

先に紹介したクイックソートのプログラムは、配列の要素が整数や実数であった場合のプログラムをそのまま構造体型のデータに対して応用したものとなっています。もちろんこの方法でソーティングを行うことは可能なわけですが、この Record 型のように 1 つの要素のデータの大きさが大きい場合[1]は、要素を交換する際のメモリのコピーの量が大きくなってしまい、全体としてソーティングプログラムの効率が悪く (ソーティングに時間がかかるように) なってしまいます。

[1]たとえば int 型のデータメモリ中での大きさが 4 byte であったとすると、prog6-1.c の Record 型のデータ 1 個の (メモリ中での) 大きさは 104 byte になります。 C のプログラムでは、たった 3 回の代入によって、要素の交換が実現されていますが、これらが実行されると、コンピュータの中で 312 byte 分のデータの転送を引き起します。

要素の交換をもっと効率よく行うために、配列中の構造体のデータそのものを交換するのではなく、構造体のデータ辺のポインタを交換して全体を整列させる方法がよく用いられます。この時、prog6-1.c と同じことを次のような方針で実現することになります。

実現の方針 (1) Record 型の配列 data を用意しておき、ここにキーボードから入力されたデータを読み込む。

(2) この配列 data と同じ長さ (要素数) を持つ Record 型へのポインタ型の配列 dp を用意しておき、各 dp[i] を data[i] のアドレスで初期化する。

(3) 配列 dp 中のアドレス (Record 型へのポインタ型) を、そのアドレスが指す構造体の score の部分が昇順となるように、クイックソートアルゴリズムで整列する。

(4) 配列 dp 中のアドレス (Record 型へのポインタ型) を、先頭から順に用いて、そのアドレスが指す構造体の score と name の部分を 1 行に 1 組ずつ出力する。

この実現の方針の (3) と (4) は、それぞれ次のように宣言できる関数 quicksort と showdata で実現することになります。

```
void quicksort(Record *d[], int h, int t);
void showdata(Record *d[], int num);
```

関数 quicksort がクイックソート法で整列を行う過程で交換されるのは (100 byte を越えるような) Record 型のデータではなく、単なるアドレス (Record 型へのポインタ型のデータ) ですから、通常の整数や実数のデータを交換するのと同程度のメモリのコピーで済むようになるわけです。

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。prog6-1.c については教員か TA のチェックが必要です。

3.1 prog6-1.c

「1.2 実力チェック」で作成した関数 quicksort を用いて、「1. 構造体の整列」の節のプログラム prog6-1.c を完成しなさい。

3.2 prog6-2.c

まず、prog6-1.c を prog6-2.c にコピーしなさい。つぎに、今回の「2. ポインタを利用して効率よく要素を交換する」の実現の方針にしたがって、prog6-2.c をポインタを使ったソーティングプログラムに変更しなさい。prog6-2.c は prog6-1.c と全く同じように動作しなければなりません。

余裕のある受講者は、prog6-1.c や prog6-2.c で定義した関数 quicksort を、前々回の prog4-3.c や前回の prog5-3.c と同様な方法で 10000 回テストするプログラム prog6-3.c prog6-4.c を作成してみましょう。 このとき、整列する配列の長さを 1 から 100 の間でランダムに選ぶようにしてテストにの時間を比較してみてください。

プログラミングおよび実習Ⅱ・第 6 回・終り

付録 A. 構造体について

ある人の名前と点数のように、いくつかのデータの組として構成されるひとまとまりのデータを、 C では、**構造体**と呼ばれるデータ型として扱うことができます。

構造体の宣言

構造体型は C の予約語 `struct` を使って、プログラム中でつぎのように宣言することにより利用可能になります[2]。

```
struct 構造体タグ {
    型指定1  メンバ名1;
    型指定2  メンバ名2;
    型指定3  メンバ名3;
      ⋮          ⋮
    型指定n  メンバ名n;
};
```

「構造体タグ」はここで定義する構造体に付ける名前 (識別子) で、変数名や関数名などと同じように適当に選びます[3]。構造体型のデータを構成している各部分を、その構造体の**メンバ**と呼びます。構造体の宣言の { と } で囲まれた部分で、この構造体のメンバのデータ型と、そのメンバを指し示すときに用いる名前 (メンバ名[4]) を指定します。この部分の書き方は、ちょうど変数や配列の宣言と同じ形になっていることに注意してください。

たとえば、年月日の 3 つの情報で構成される日付を表すデータは型は、

```
struct DATE {
    int year;       /* 年 */
    int month;      /* 月 */
    int day;        /* 日 */
};
```

のように宣言される構造体として表現できるでしょう。

構造体の宣言はあくまでデータ型の宣言ですから、これで何らかの変数や配列が準備されるわけではないことに注意してください。構造体型のデータを記憶するためには (次節で紹介するように) その構造体型の変数 (や配列) を宣言 (定義) して、それを利用することになります。

構造体型の変数宣言

構造体型も 1 つのデータ型ですから[5]、その型のデータを変数に記憶することができます。構造体型のデータを記憶するための変数は、その変数のデータ型を「`struct` 構造体タグ」の形式で指定して宣言します。たとえば、前節のように `DATE` という構造体が宣言されている場合、

```
struct DATE today, tomorrow;
```

---

[2]最後の ; を忘れないように注意しましょう。構造体の宣言は、関数定義の外やブロックの始まりなど、変数の宣言を行うことができる場所に書くことができます。構造体の宣言が有効となる範囲も変数の宣言の場合と同様です

[3]構造体タグは他の変数や関数の名前と重なっても構いません

[4]変数名や関数名などと同様な名前を適当に選んぶことができます。メンバ名についても他の変数や関数と同じ名前を使って構いません。また、異なる構造体が同じ名前のメンバを持っていても区別されます

[5]当然、構造体型の要素からなる配列や、別の構造体型のメンバを持つ構造体型などのデータ型を使うことができます

のように、この構造体型のデータを記憶する変数 today や tomorrow を宣言 (定義) することができます。これらはちょうど、int 型の変数 x、y の宣言

```
int x, y;
```

の int が、struct DATE に置き換わった形になっていることに注意してください。

変数の宣言は、構造体の宣言と一緒にして、

```
struct DATE {
    int year;       /* 年 */
    int month;      /* 月 */
    int day;        /* 日 */
} today, tommorow;
```

のようにすることもできます[6]。

構造体型の変数の初期化

構造体型の変数を宣言と同時に初期化するためには、次の例のように変数名に続いて、= と、各メンバの初期値を , で区切って { } の中に書きます[7]。

```
struct DATE today = { 2003, 5, 28 };
```

この {} を使った形式の値の設定は、変数の宣言時にしか使用することはできません。たとえば、

```
struct DATE today;
```

のように変数 today を宣言した後に、

```
today = { 2003, 5, 28 };
```

と書いて、この変数の各メンバに代入を行うことは許されませんので注意が必要です。次節に紹介する方法で、構造体のメンバそれぞれに対して個別に値を代入します。

構造体メンバへのアクセス

構造体の各メンバにアクセスするためには、**直接メンバ演算子** “.” を用います。s がある構造体型のデータ (を表す式) で、m がその構造体のメンバ名のとき

$$s.m$$

という式は、s の m というメンバの値を意味します。s が変数や配列要素などの代入可能な対象[8]である場合は、この式を代入演算子 = の左辺に書いて、s の m というメンバだけを変更することもできます。たとえば、先のように宣言された変数 today と tommorow があった場合、

```
tomorrrow.day = today.day;
tomorrrow.day++;
```

のようなプログラムを書くことが可能です。

---

[6] この場合、これらの構造体の定義を他の場所で使う予定がないのであれば、構造体タグの「DATE」は省略しても構いません

[7] 配列を初期化する場合も {} の中に配列要素の初期値を並べて書きますから、配列をメンバとする構造体初期化では、{} が入れ子となる場合があります

[8] 代入演算子の左辺に書けるようなものを**左辺値**と呼びます。変数や配列の要素が左辺値の代表的な例です。左辺値である構造体の各メンバはやはり左辺値となります

構造体全体のコピー

次のプログラム tomorrow1.c のように、構造体型のデータ全体を変数に代入したり、関数の実引数として関数に渡したり、関数の返り値として呼び出し元に返したりすることも可能です。いずれの場合も、その構造体を構成しているすべてのデータ (メンバ) のコピーが起きます。

tomorrow1.c

```
 1 #include <stdio.h>
 2
 3 struct DATE {
 4     int year;
 5     int month;
 6     int day;
 7 };
 8
 9 #define IS_LEAP_YEAR(y) ((y)%4 == 0 && ((y)%100 != 0 || y %400 == 0))
10
11 /* 引数 d の次の日を返す */
12 struct DATE nextDay(struct DATE d)
13 {
14     /* 各月の日数 */
15     int dom[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
16
17     if (IS_LEAP_YEAR(d.year))                /* うるう年か? */
18         dom[1]++;
19     if (++d.day > dom[d.month - 1]) {        /* 次の月になるか? */
20         d.day = 1;
21         if (++d.month > 12) {                /* 次の年になるか? */
22             d.month = 1;
23             d.year++;
24         }
25     }
26     return d;
27 }
28
29 int main()
30 {
31     struct DATE today = { 2003, 10, 27 }, tomorrow;
32
33     tomorrow = nextDay(today);
34     printf ("今日は西暦%d年%d月%d日です\n",
35             today.year, today.month, today.day);
36     printf ("明日は西暦%d年%d月%d日です\n",
37             tomorrow.year, tomorrow.month, tomorrow.day);
38     return 0;
39 }
```

構造体へのポインタ

C では、配列全体を一度に代入したり、関数の引数や返り値として配列全体をやり取りすることはできませんが、構造体のメンバの 1 つとして配列を含ませておけば、この構造体を利用することで (実質的に) 配列全体の代入や、関数間での配列全体のやり取りも可能となります。だだし、その構造体に含まれるすべてのデータのコピーが起りますので、構造体そのものをコピーしながら取り扱うのではなく、構造体へのポインタ値 (アドレス) を取り扱う方が効率的な場合が多くあります。これは、構造体のメンバに配列が含まれていない場合でも同様です。

構造体型へのポインタ (構造体型へのポインタ型の変数) は、他のデータ型へのポインタと同様に、つ

ぎの例のように宣言することができます。

```
struct DATE *p, *q;
```

関数の仮引数となる場合も同様です。

p が構造体へのポインタ型の値 (アドレス) を表す式で、 m がその構造体のメンバ名のとき、

$$p\texttt{->}m$$

という式で p が指す構造体 (アドレス p に格納されている構造体) のメンバ m にアクセスすることができます。この "->" は**間接メンバ演算子**と呼ばれます。「p->m」という式は「(*p).m」と同じ意味となることに注意してください。

次のプログラム tommorow2.c は、先に紹介した tomorrow1.c の関数 nextDay を、引数として渡したアドレスの指す DATE 構造体の値を 1 日進めるようなものに変更して、これを使って元のプログラムと同じ働きをさせるようにしたものです[9]。

tomorrow2.c

```
 1 #include <stdio.h>
 2
 3 struct DATE {
 4     int year;
 5     int month;
 6     int day;
 7 };
 8
 9 #define IS_LEAP_YEAR(y) ((y)%4 == 0 && ((y)%100 != 0 || y %400 == 0))
10
11 /* 引数 d を次の日に進める */
*12 void nextDay(struct DATE *d)
13 {
14     /* 各月の日数 */
15     int dom[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
16
*17     if (IS_LEAP_YEAR(d->year))          /* うるう年か? */
18         dom[1]++;
*19     if (++d->day > dom[d->month - 1]) { /* 次の月になるか? */
*20         d->day = 1;
*21         if (++d->month > 12) {          /* 次の年になるか? */
*22             d->month = 1;
*23             d->year++;
24         }
25     }
*26 }
27
28 int main()
29 {
30     struct DATE today = { 2003, 5, 28 }, tomorrow;
31
*32     tomorrow = today;
*33     nextDay(&tomorrow);
34     printf ("今日は西暦%d年%d月%d日です\n",
35             today.year, today.month, today.day);
36     printf ("明日は西暦%d年%d月%d日です\n",
37             tomorrow.year, tomorrow.month, tomorrow.day);
38     return 0;
39 }
```

[9]書き換えた行に * 印が付してあります

付録 B. 前回の演習問題 prog5-2.c のプログラム例

prog5-2.c

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 typedef int Boolean;
 5 #define TRUE            (1)
 6 #define FALSE           (0)
 7
 8 #define MAX_DATA        1000
 9
10 void showdata (double data[], int num)
11 {
12     int i;
13
14     if (num > 0) {
15         printf("%6.2f", data[0]);
16         for (i = 1; i < num; i++)
17             printf(",%6.2f", data[i]);
18     }
19     printf("\n");
20 }
21
22 void quicksort(double d[], int h, int t)
23 {
24     double r, tmp;
25     int i, j;
26
27     if (h >= t)
28         return;
29     r = d[(h+t)/2];
30     i = h;
31     j = t;
32     while (1) {
33         while (d[i] < r)
34             i++;
35         while (r < d[j])
36             j--;
37         if (j <= i)
38             break;
39         tmp = d[i];
40         d[i++] = d[j];
41         d[j--] = tmp;
42     }
43     quicksort(d, h, i-1);
44     quicksort(d, j+1, t);
45 }
46
47 int main ()
48 {
49     double 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 ("%lf", &data[num]) != 1)
55             break;
56     }
57     quicksort(data, 0, num-1);
58     showdata(data, num);
59     return EXIT_SUCCESS;
60 }
```