プログラミングおよび実習II・第 7 回

2003 年 11 月 5 日

1　実力チェック

次のプログラムは構造体 Record 型へのポインタ型のデータを、その指す先に置かれている構造体データの score の部分が昇順となるようにクイックソート法でソーティングする関数 quicksort の定義です[1]。

この関数 quicksort が (引数の h と t で) 与えれた区間を整列するために行った基準値 r と点数 (score) の比較 (プログラムの 14 行目や 16 行目) の回数を、int 型の返り値として返すように、その定義を変更しなさい。ただし、関数 quicksort が返す回数としては、そこから再帰的に呼び出された側で行われた (基準値と点数の) 比較の回数もすべて含めて返すようにしなさい。解答用紙のプログラム名は「prog7-1.c」とし、関数 quicksort の (変更後の) 定義のみを書きなさい。

```
 1 void quicksort(Record *d[], int h, int t)
 2 {
 3     Record *tmp;
 4     int r;
 5     int i, j;
 6
 7     if (h >= t)
 8         return;
 9
10     r = d[(h+t)/2]->score;
11     i = h;
12     j = t;
13     while (1) {
14         while (d[i]->score < r)
15             i++;
16         while (r < d[j]->score)
17             j--;
18         if (j <= i)
19             break;
20         tmp = d[i];
21         d[i++] = d[j];
22         d[j--] = tmp;
23     }
24     quicksort(d, h, i-1);
25     quicksort(d, j+1, t);
26 }
```

2　整列のために要した時間を計測する

前回、名前 (100 個の要素からなる char 型配列) と点数 (int 型) の 2 つのデータで構成される構造体を考え、この構造体型のデータの配列をそのまま整列するプログラム prog6-1.c と、構造体へのポインタ型の配列を整列するプログラム prog6-2.c を作成しました。プログラム prog6-1.c を prog6-2.c へ変更した理由は、整列中に配列の要素が交換される際にコピーされるデータの量を減らして、プログ

[1]「付録 B. 前回の演習問題 prog6-2.c のプログラム例」を参照してください。

ラムを効率化するためでしたが、この変更でプログラムはどの程度高速に整列を行うことができるようになったのでしょうか?

1-542 や 1-609 実習室の Linux 環境では、/home/sample/mprog2/util.h というファイルにプログラムの実行時間を測定するための関数群が用意されています。これを使って、prog6-2.c でどのくらいプログラムが効率化されたのかを測定してみましょう。util.h には次のような関数の定義が含まれています。ただし、これらの関数では 100 分の 1 秒単位でしか時間を計測することができませんので注意が必要です。

`void clock_on(void);`	時間測定用の時計を動かし始める
`double clock_time(void);`	現在までの累計時間 (単位は秒) を double 型の値で返す
`double clock_off(void);`	時間測定用の時計を止めて、その時までの累計時間 (単位は秒) を double 型の値で返す
`void clock_reset(void);`	時間測定用の時計を 0 秒に戻す

これらの関数を利用する場合は、/home/sample/mprog2/util.h を Prog2 ディレクトリにコピーしておき、これらの関数を呼び出すプログラムのソースファイルの冒頭に次の 1 行を加えます。

```
#include "util.h"
```

たとえば、付録 B の prog6-2.c での関数 quicksort の実行に費される時間を計測して表示する場合は、次のプログラム prog7-2.c のようにします[2]。

とりあえずの prog7-2.c

```
  #include <stdio.h>
  #include <stdlib.h>
* #include "util.h"
  :
  int main ()
  {
      Record data[MAX_DATA], *dp[MAX_DATA];
*     int num, i;

      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;
      }
*     clock_on();
*     for (i = 0; i < num; i++)
*         dp[i] = &data[i];
      quicksort(dp, 0, num-1);
*     printf("整列に要した時間 = %.3f 秒\n", clock_off());
      return EXIT_SUCCESS;
  }
```

整列結果の表示はとりあえず必要ないので、showdata の呼び出しは削除してあります。また、配列 dp の初期化は、構造体の配列をそのまま整列する場合 (prog6-1.c) にはなかった処理ですから、公平を期するために、この初期化に要した時間も含めて計測するようにしました。このプログラムをコンパイルして実行してみると次のようになります。

[2] prog6-2.c からの変更部分には * 印が付してあります

```
s1542h017% cc -o prog7-2 prog7-2.c
s1542h017% ./prog7-2
 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 => .
整列に要した時間 = 0.000 秒
s1542h017%
```

「整列に要した時間 = 0.000 秒」と表示されているのは、この例のようにデータの数が少ない場合、整列に必要な時間が短かすぎて util.h が使っている時計 (100 分の 1 秒ごとに針が進む) では正確に計測することができないからです。

まずは、もう少し大量のデータを整列させることを考えましょう。配列中の要素数の最大 (MAX_DATA) を 100000 に変更して、この数のデータを整列させてみます。人間の手で 100000 個のデータを入力するのは大変ですから、整列プログラムをテストした時 (第 4 回の 5 ページ) のように乱数を使って入力データを生成することにします。このときの関数 main の定義は次のようになります。

要素数を 100000 にした prog7-2.c

```
  #include <stdio.h>
  #include <stdlib.h>
* #include <string.h>
  #include "util.h"
  ⋮
* #define MAX_DATA      (100000)
  ⋮
  int main ()
  {
*     static Record data[MAX_DATA], *dp[MAX_DATA];
*     int num, i;

*     num = MAX_DATA;
*     for (i = 0; i < num; i++) {
*         /* 点数を乱数で生成する */
*         data[i].score = rand();
*         /* 名前は "Test" に固定 */
*         strcpy(data[i].name, "Test");
*     }
*     clock_on();
*     for (i = 0; i < num; i++)
*         dp[i] = &data[i];
*     quicksort(dp, 0, num-1);
*     printf("整列に要した時間 = %.3f 秒\n", clock_off());
      return EXIT_SUCCESS;
  }
```

要素数を 100000 にしたことで、配列 data の大きさは 10MByte 程度になってしまいますので、関数 main の中でのこの配列の宣言には static が追加されています。static がない宣言の場合、変数や配

列のデータを記憶するためのメモリは、関数 main が呼ばれた時点で**スタック**と呼ばれる領域[3]に確保されますが、このスタック領域の大きさは (通常は) 比較的小さな量に制限されているため[4]、大きな配列を割り当てることができない場合があります[5]。一方、変数や配列を static 宣言すると、その変数や配列は、ブロック (関数) の実行が始まる度に (スタック領域に動的に) 割り当てられるのではなく、プログラムの実行が開始される時にスタック領域とは別のメモリ領域に (1 回だけ) 割り当てられます。この領域はスタック領域に比べると制限が (通常は) 非常に緩くなっていますので、大量のメモリを割り当てることが可能です[6]。

この配列 data の 100000 個の要素は、それぞれ構造体 (Record 型) になっているわけですが、その点数の部分 (score) は rand() が返す値で、名前の部分は常に同じ文字列 "Test" で初期化しています。名前の初期化に使われている関数 strcpy は、

```
char *strcpy(char *s1, const char *s2);
```

のように宣言される標準ライブラリ関数です。この関数を利用する際には、プログラムの冒頭部分に

```
#include <string.h>
```

の 1 行を書いておきます。関数 strcpy は、s2 で指定された文字列を、s1 で指定されたアドレスを先頭として (終端の 0 を含めて) コピーします[7]。関数 strcmp の宣言に使われている const は C の予約語 (キーワード) の 1 つで、const char *s2 と仮引数 s2 を宣言することで、ポインタ型変数 s2 の指す先 (アドレス) に置かれている char 型のデータが**変更されない**ことを示しています。関数 strcpy は s2 を s1 へコピーするわけですが、s2 の方の文字列の内容は、関数 strcmp によって参照されることはあっても変更されることはありません。const はこのことを明示する働きを持っています。

この prog7-2.c というプログラムでは、関数 quicksort は 100000 個の構造体データを整列することになりますが、それでも実行時間は 1 秒に満たないものです。これでは 2 つの方法の実行効率を正確に比較することができなくなってしまいます。util.h で用意されている関数群は 100 分の 1 秒単位でしか時間を計ることができませんから、たとえば $\pm 3\%$ 以下の誤差で実験結果を得たい場合、最低でも 3分の1 秒 (= 0.01秒/0.03) 以上の (通常は余裕をみてその 10 倍以上の) 累計時間となるようにしなければなりません。そこで、何回も整列を繰り返し行った時に要した時間を計測して累積時間が大きくなるようにし、これを整列を繰り返した回数で割ることで 1 回の整列に要した時間を算出することにします。次のプログラム (関数 main の定義) では、このような方針で 100000 個の要素からなる配列を整列するのに必要な時間を求めて出力しています。

最終的な prog7-2.c の関数 main の定義

```
#define REPEAT        (30)

int main()
```

[3]スタック領域には、関数の仮引数の値、その関数のリターンアドレス (その関数を呼び出した側で、どのアドレスから機械語プログラムを続行すべきかに関する情報)、static 宣言されていない局所変数の値、複雑な式を計算する際の部分的な計算結果などが記憶されます。

[4]たとえば、8MByte までに制限されています。

[5]プログラムを実行した際に「セグメントエラー」、「セグメント違反」、「セグメンテーションフォルト」などと呼ばれる現象が起きてプログラムは異常終了してしまいます。

[6]たとえば、無制限 (そのコンピュータの能力の限界まで大きくなることができるよう) に設定されています。

[7]strcpy の返り値は、引数 s1 の値そのものですので、あまり役に立ちません。

```
{
    static Record data[MAX_DATA], *dp[MAX_DATA];
    int num, i, j;
    double t1, t2;

    num = MAX_DATA;
    /* まずデータ生成のためだけに必要な時間を求めておく */
    clock_on();
    for (j = 0; j < REPEAT; j++) {
        /* 点数は乱数、名前は "Test" に固定してデータを生成する */
        for (i = 0; i < num; i++) {
            data[i].score = rand();
            strcpy(data[i].name, "Test");
        }
    }
    t1 = clock_off();

    /* 次にデータ生成と整列に必要な時間を求める */
    clock_reset();
    clock_on();
    for (j = 0; j < REPEAT; j++) {
        /* データを生成する */
        for (i = 0; i < num; i++) {
            data[i].score = rand();
            strcpy(data[i].name, "Test");
        }
        /* データを整列する */
        for (i = 0; i < num; i++)
            dp[i] = &data[i];
        quicksort(dp, 0, num-1);
    }
    t2 = clock_off();

    /* 整列1回当りの時間を表示する */
    printf("整列に要した時間 (%.3f - %.3f) / %d = %.5f 秒\n",
           t2, t1, REPEAT, (t2 - t1) / REPEAT);
    return EXIT_SUCCESS;
}
```

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。prog7-1.c については教員か TA のチェックが必要です。

3.1 prog7-1.c

「1. 実力チェック」の節で作成した関数 quicksort を用いて、次の実行例のように比較の総数を出力するプログラム prog7-1.c を完成しなさい。

prog7-1.c の実行例

```
s1542h017% ./prog7-1
 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
比較の総数 = 32
s1542h017%
```

3.2 prog7-2.c

まず、前回の演習問題で作成した「ポインタを利用して効率よく」クイックソートを行うプログラム prog6-2.c を prog7-2.c にコピーしなさい。つぎに「2. 整列のために要した時間を計測する」の節を参考にして、このプログラムで 100000 個の要素を整列するのに要した (平均) 時間を測定して、次の実行例のように表示するプログラムに変更しなさい。ただし、測定誤差が $\pm 1\%$ 未満になるように整列の回数を調節しなさい。

prog7-2.c の実行例

```
s1542h017% ./prog7-2
整列に要した時間 (15.660 - 2.110) / 100 = 0.13550 秒
s1542h017%
```

余裕のある受講者は、構造体そのものを整列するプログラム prog6-1.c についても、整列に要した時間を計測して表示するプログラム prog7-3.c を作成して、整列時間を prog7-2.c の場合と比較してみて下さい。

プログラミングおよび実習Ⅱ・第 7 回・終り

付録 A. 前回の演習問題 prog6-1.c のプログラム例

prog6-1.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 NAME_LEN        (100)
 9 #define MAX_DATA        (1000)
```

```
10
11 typedef struct {
12     char name[NAME_LEN];
13     int score;
14 } Record;
15
16 void showdata(Record d[], int num)
17 {
18     int i;
19
20     for (i = 0; i < num; i++)
21         printf("%3d %s\n", d[i].score, d[i].name);
22 }
23
24 void quicksort(Record d[], int h, int t)
25 {
26     Record tmp;
27     int r;
28     int i, j;
29
30     if (h >= t)
31         return;
32
33     r = d[(h+t)/2].score;
34     i = h;
35     j = t;
36     while (1) {
37         while (d[i].score < r)
38             i++;
39         while (r < d[j].score)
40             j--;
41         if (j <= i)
42             break;
43         tmp = d[i];
44         d[i++] = d[j];
45         d[j--] = tmp;
46     }
47     quicksort(d, h, i-1);
48     quicksort(d, j+1, t);
49 }
50
51 int main ()
52 {
53     Record data[MAX_DATA];
54     int num;
55
56     for (num = 0; num < MAX_DATA; num++) {
57         printf("%2d番目の点数と名前 => ", num+1);
58         if (scanf("%d %99[^\n]", &data[num].score, data[num].name) != 2)
59             break;
60     }
61     quicksort(data, 0, num-1);
62     showdata(data, num);
63     return EXIT_SUCCESS;
64 }
```

付録 B. 前回の演習問題 prog6-2.c のプログラム例

prog6-2.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 NAME_LEN        (100)
 9 #define MAX_DATA        (1000)
10
11 typedef struct {
12     char name[NAME_LEN];
13     int score;
14 } Record;
15
16 void showdata(Record *d[], int num)
17 {
18     int i;
19
20     for (i = 0; i < num; i++)
21         printf("%3d %s\n", d[i]->score, d[i]->name);
22 }
23
24 void quicksort(Record *d[], int h, int t)
25 {
26     Record *tmp;
27     int r;
28     int i, j;
29
30     if (h >= t)
31         return;
32
33     r = d[(h+t)/2]->score;
34     i = h;
35     j = t;
36     while (1) {
37         while (d[i]->score < r)
38             i++;
39         while (r < d[j]->score)
40             j--;
41         if (j <= i)
42             break;
43         tmp = d[i];
44         d[i++] = d[j];
45         d[j--] = tmp;
46     }
47     quicksort(d, h, i-1);
48     quicksort(d, j+1, t);
49 }
50
51 int main()
52 {
53     Record data[MAX_DATA], *dp[MAX_DATA];
54     int num;
55
56     for (num = 0; num < MAX_DATA; num++) {
57         printf("%2d番目の点数と名前 => ", num+1);
58         if (scanf("%d %99[^\n]", &data[num].score, data[num].name) != 2)
59             break;
60         dp[num] = &data[num];
61     }
62     quicksort(dp, 0, num-1);
63     showdata(dp, num);
64     return EXIT_SUCCESS;
65 }
```