プログラミングおよび実習II・第 13 回

2003 年 12 月 17 日

1　実力チェック

double 型の要素からなる配列 d と、その要素数 n (int 型)、検索対象となるデータ x (double 型)、検索を始める添字 start (int 型) の 4 つを引数として受け取り、配列 d の添字 start の要素から始めて (末尾に向って) x と等しい要素を (線形探索法で) 探し、そのような要素が見つかった場合は、最初に見つかった要素へのポインタを返り値として返す関数 dsearch を定義しなさい。ただし、配列を末尾まで探しても見つからなかった場合は、配列の先頭から探索を続けるようにし、すべての要素を調べても見つからない場合には NULL (ポインタ型の 0) を返り値として戻すようにしなさい。解答用紙のプログラム名は「dsearch.c」とし、関数 dsearch の定義のみを書きなさい。ただし、マクロ NULL をプログラム中で使用するために、プログラムの冒頭には「#include <stdlib.h>」を書くようにしなさい。

2　ハッシュ法を用いて郵便番号から住所を検索する

前々回と前回では、郵便番号の一覧が格納されたファイル /home/sample/mprog2/zipcode から読み込まれたデータに対して、線形探索法や二分探索法を使って郵便番号の検索を行うプログラムを作成しましたが、今回は、ハッシュ法[1]を用いて同じことを行うプログラム prog13-1.c を作ってみましょう。このプログラムの実行例も前々回の prog11-1.c と同様になるはずです。ただし、入力された郵便番号を持つ行が複数ある場合に、どの行が出力されるかは予想が付かない点は二分探索法と同様となります。

実現の方針

(1) 前々回の prog11-1.c と同様に、ファイル /home/sample/mprog2/zipinfo から読み取ったデータは ZipInfo 型の配列 data に格納しておく。また、読み取ったデータの件数を int 型の変数 num に格納する。

(2) ファイルの内容を読み込む配列 data とは別に、ZipInfo 型へのポインタ型の配列 hashtable を用意しておき、これをハッシュ表として用いる。値が NULL (ポインタ型の 0) である要素は、ハッシュ表のその欄が使われていないことを示すものとする。読み込まれるデータの総数は 12 万件余なので、(目安として) その倍以上の大きさ (たとえば 300000) を確保しておく。ハッシュ表の大きさが十分でないとハッシュ値の衝突が頻繁に起きてしまい探索の効率が悪化する。

(3) ハッシュ表を空にするために、配列 hashtable の各要素を NULL (ポインタ型の 0) で初期化する。配列 hashtable を static という修飾子を付けて宣言しておけば自動的に NULL で初期化されるので、明示的に NULL を代入する必要はないことに注意する。

(4) ファイルの内容から 1 件のデータを配列 data に読み込む度に、読み込んだ要素へのポインタをハッシュ表 hashtbale に登録する。この登録作業は、次のような関数 hashadd を定義しておき、この関数を呼び出して行うようにする。

[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 164 ページからを適宜参照してください。

```
int hashadd(ZipInfo *htable[], int tsize, ZipInfo *p);
```

(4-1) 関数 hashadd は、ハッシュ表 htable とその大きさ tsize、登録したい ZipInfo 型のデータへのポインタ p を引数として受け取る。

(4-2) p->code の値から適当なハッシュ関数を使ってハッシュ値 h を計算する。h は、配列 htable の添字の範囲 (0...tsize−1) に収まるようなものでなければならない。

(4-3) ハッシュ関数としては

```
#define HASH(c, s) ((c)*43%(s))
```

のような関数形式のマクロを定義しておいて、

```
h = HASH(p->code, tsize);
```

のようにして p->code に対するハッシュ値を計算すればよい。「43」の部分は、ハッシュ表の大きさや 10000 と互に素な正の整数を選ぶようにするとよい。郵便番号の下 4 桁 (0...9999) は特定の数値 (たとえば 100 未満) である場合が多いので、(たとえば) ハッシュ表の大きさ tsize が 300000 の時、もし

```
#define HASH(c, s) ((c)%(s))
```

のようにハッシュ関数を定義してしまうと、計算されるハッシュ値に偏りが生じ、ハッシュ値の衝突が頻出すると予想される。このため、ハッシュ表の大きさや 10000 と互に素な正の整数 k (この例では 43) を選んで、郵便番号をまず k 倍した後、ハッシュ表の大きさで割った時の余りを求めて、それをハッシュ値として採用している[2]。

(4-4) 配列 htable の添字 h の要素が NULL (その欄が空き) であれば、p の値をその要素に格納する。NULL でなければ、空きが見つかるまで、添字 h を 1 増加させて、空きを見つけ、そこに p を格納する[3]。配列の末尾まで探しても空きが見つからない場合は、配列の先頭に戻って空きを探す。空きが見つかって、そこに p の値を格納できた場合は、その添字 h をこの関数の返り値として返す。すべての要素を探しても空きが無かった場合は、登録をあきらめて -1 を返り値として返す。

(5) 探索すべき郵便番号 (整数) を標準入力 (キーボード) から scanf を使って読み取り、ハッシュ表 htable を使ったハッシュ法による探索を行って得られた情報を標準出力 (端末ウィンドウ) に出力する。関数 main は scanf が整数 (郵便番号) の読

[2] k が大きいと p->code を k 倍した計算結果が int 型のデータとして表現できなくなって (オーバーフローして) しまい、ハッシュ値が予期できない値 (たとえば負の値) となってしまうことがあるので注意が必要です。

[3] このように、ハッシュ表の別の場所 (この例では次の添字) に空きを探して登録することでハッシュ値の衝突を処理する方法を**オープンアドレス法**と呼びます。

み込みに失敗するまで、これを繰り返す。この時、次のような関数 hashsearch を定義しておき、この関数を呼び出してハッシュ法による探索を行う。

```
ZipInfo *hashsearch(ZipInfo *htable[], int tsize, int code);
```

(5-1) 関数 hashsearch は、ハッシュ表 htable とその大きさ tsize、探索したい郵便番号 code を引数として受け取る。

(5-2) code から (関数 hashadd が使ったものと同じ) ハッシュ関数を使ってハッシュ値 h を計算する。

(5-3) 関数 hashsearch は、配列 htable の添字 h の要素から順に (線形探索法と同様に) 探索して、htable の要素が指す先の ZipInfo 型のデータの code メンバが引数の code と等しいものを見つける。見つかった場合は、その ZipInfo 型のデータへのポインタ (つまり、ハッシュ表のある要素の値) を関数の返り値として返す。htable の末尾まで探しても見つからなかった場合は、htable の先頭に戻って探索を続けるが、探索中にハッシュ表の空き (要素の値が NULL) に至った場合 (および、ハッシュ表をすべて探索してしまった場合) は、探索すべきデータは存在しないものとして NULL を返す。

prog13-1.c の関数 main の定義は以下のようなものとなります。

prog13-1.c の関数 main の定義

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 #define FILE_NAME        "/home/sample/mprog2/zipcode"
 5
 6 #define MAX_DATA        (200000)
 7 #define PREF_NAME_LEN   (20)
 8 #define CITY_NAME_LEN   (40)
 9 #define AREA_NAME_LEN   (100)
10
11 typedef struct {
12     int code;                    /* 郵便番号 */
13     char pref[PREF_NAME_LEN];    /* 都道府県名 */
14     char city[CITY_NAME_LEN];    /* 市町村名 */
15     char area[AREA_NAME_LEN];    /* 町域名 */
16 } ZipInfo;
17
18 #define HASH_TABLE_SIZE        (300000)
19
20 int hashadd(ZipInfo *htable[], int tsize, ZipInfo *p);
21 ZipInfo *hashsearch(ZipInfo *htable[], int tsize, int code);
22
23 int main()
24 {
25     static ZipInfo data[MAX_DATA], *hashtable[HASH_TABLE_SIZE];
26     FILE *fp;
27     int code;
28     int num;
29     ZipInfo *p;
30
31     fp = fopen(FILE_NAME, "r");
32     if (fp == NULL) {
```

```
33          printf("%s というファイルが読めません\n", FILE_NAME);
34          exit(EXIT_FAILURE);
35      }
36
37      num = 0;
38      while (fscanf(fp, "%d %s %s %s", &data[num].code,
39                      data[num].pref, data[num].city, data[num].area) == 4) {
40          if (hashadd(hashtable, HASH_TABLE_SIZE, &data[num]) < 0) {
41              printf("ハッシュ表が溢れました\n");
42              fclose(fp);
43              exit(EXIT_FAILURE);
44          }
45          num++;
46      }
47
48      fclose(fp);
49      printf("%s から %d 件のデータを読み込みました\n", FILE_NAME, num);
50
51      while (1) {
52          printf ("郵便番号を入力してください : ");
53          if (scanf("%d", &code) != 1)
54              break;
55          p = hashsearch(hashtable, HASH_TABLE_SIZE, code);
56          if (p == NULL)
57              printf("対応する地域がありません。\n");
58          else
59              printf("%07d %s %s %s\n", p->code, p->pref, p->city, p->area);
60      }
61      printf("検索を終了します。\n");
62      return EXIT_SUCCESS;
63  }
```

3 演習問題

次の演習問題に取り組み、prog13-1.c については自分が作成したプログラムを mprog2 コマンドで提出してください。教員か TA によるチェックも必要です。

3.1 prog13-1.c

「2. ハッシュ法を用いて郵便番号から住所を検索する」の節で解説した実現の方針に従って、プログラム prog13-1.c を完成しなさい。プログラムが完成したら mprog2 コマンドで提出してください。教員か TA によるチェックも必要となります。関数 hashadd や関数 hashsearch をこのような名前で定義しないと mprog2 コマンドが受理しませんので注意して下さい。

3.2 prog13-2.c

余裕のある受講者は、前回の prog12-2.c や prog12-3.c と同様にハッシュ法を使って 1 回の探索を行うときの所要時間を調べて表示するようなプログラム prog13-2.c を作成してみましょう。この問題のプログラムは mprog2 コマンドで提出する必要はありません。ファイルから読み込んだデータのハッシュ表への登録が完了した状態で、1 回の探索に必要な時間を計測します。

次は prog13-2.cの実行例です。

prog13-2.c の実行例

```
s1542h017% ./prog13-2 10
10件からの探索に要した時間（49.190 - 39.390）/ 25600871 = 0.0000003828 秒
s1542h017% ./prog13-2 100
100件からの探索に要した時間（49.190 - 39.380）/ 25594900 = 0.0000003833 秒
s1542h017% ./prog13-2 1000
1000件からの探索に要した時間（48.390 - 38.650）/ 25119398 = 0.0000003877 秒
s1542h017% ./prog13-2 10000
10000件からの探索に要した時間（46.160 - 36.400）/ 23651378 = 0.0000004127 秒
s1542h017% ./prog13-2 100000
100000件からの探索に要した時間（28.850 - 18.980）/ 12333325 = 0.0000008003 秒
s1542h017%
```

ハッシュ表の大きさやハッシュ関数の定義をいろいろと変えてみて、探索時間がどのように変化するか調べて見ましょう。

プログラミングおよび実習Ⅱ・第 13 回・終り

付録. 前回の演習問題の解答例

prog12-1.c

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 #define FILE_NAME        "/home/sample/mprog2/zipcode"
 5
 6 #define MAX_DATA        (200000)
 7 #define PREF_NAME_LEN   (20)
 8 #define CITY_NAME_LEN   (40)
 9 #define AREA_NAME_LEN   (100)
10
11 typedef struct {
12     int code;                    /* 郵便番号 */
13     char pref[PREF_NAME_LEN];    /* 都道府県名 */
14     char city[CITY_NAME_LEN];    /* 市町村名 */
15     char area[AREA_NAME_LEN];    /* 町域名 */
16 } ZipInfo;
17
18 void quicksort(ZipInfo *d[], int h, int t)
19 {
20     ZipInfo *tmp;
21     int r;
22     int i, j;
23
24     if (h >= t)
25         return;
26
27     r = d[(h+t)/2]->code;
28     i = h;
29     j = t;
30     while (1) {
31         while (d[i]->code < r)
32             i++;
33         while (r < d[j]->code)
34             j--;
35         if (j <= i)
36             break;
37         tmp = d[i];
38         d[i++] = d[j];
39         d[j--] = tmp;
40     }
41     quicksort(d, h, i-1);
42     quicksort(d, j+1, t);
43 }
44
45 int binarysearch(ZipInfo *data[], int num, int code)
46 {
47     int h = 0, t = num-1, m;
48
49     while (h <= t) {
50         m = (h+t)/2;
51         if (code < data[m]->code)
52             t = m-1;
53         else if (code > data[m]->code)
54             h = m+1;
55         else
56             return m;
57     }
```

```
58      return -1;
59  }
60
61  int main()
62  {
63      static ZipInfo data[MAX_DATA], *dp[MAX_DATA];
64      FILE *fp;
65      int code;
66      int num, pos;
67
68      fp = fopen(FILE_NAME, "r");
69      if (fp == NULL) {
70          printf("%s というファイルが読めません\n", FILE_NAME);
71          exit(EXIT_FAILURE);
72      }
73
74      num = 0;
75      while (fscanf(fp, "%d %s %s %s", &data[num].code,
76                    data[num].pref, data[num].city, data[num].area) == 4) {
77          dp[num] = &data[num];
78          num ++;
79      }
80
81      fclose(fp);
82      printf("%s から %d 件のデータを読み込みました\n", FILE_NAME, num);
83
84      quicksort(dp, 0, num-1);
85      while (1) {
86          printf ("郵便番号を入力してください : ");
87          if (scanf("%d", &code) != 1)
88              break;
89          pos = binarysearch(dp, num, code);
90          if (pos < 0)
91              printf("対応する地域がありません。\n");
92          else
93              printf("%07d %s %s %s\n", dp[pos]->code,
94                     dp[pos]->pref, dp[pos]->city, dp[pos]->area);
95      }
96      printf("検索を終了します。\n");
97      return EXIT_SUCCESS;
98  }
```