プログラミングおよび実習II・第 11 回

2003 年 12 月 3 日

1　実力チェック

int 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 (int 型) の 3 つを引数として受け取り、配列 d の先頭から順に 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の添字を、見つからなかった場合は -1 を int 型の返り値として戻すような関数 isearch を定義しなさい。解答用紙のプログラム名は「isearch.c」とし、関数 isearch の定義のみを書きなさい。

2　線形探索法を用いて郵便番号から住所を検索する

1-542 や 1-609 実習室の Linux 環境の /home/sample/mprog2/zipcode というファイルには、日本全国の郵便番号 (121667 件) が、次のような形式で格納されています。

/home/sample/mprog2/zipcode の内容

```
0600000 北海道 札幌市中央区 以下に掲載がない場合
0640941 北海道 札幌市中央区 旭ケ丘
0600041 北海道 札幌市中央区 大通東
0600042 北海道 札幌市中央区 大通西(1〜19丁目)
0640820 北海道 札幌市中央区 大通西(20〜28丁目)
0600031 北海道 札幌市中央区 北一条東
0600001 北海道 札幌市中央区 北一条西(1〜19丁目)
0640821 北海道 札幌市中央区 北一条西(20〜28丁目)
0600032 北海道 札幌市中央区 北二条東
0600002 北海道 札幌市中央区 北二条西(1〜19丁目)
   ⋮       ⋮          ⋮          ⋮
9071431 沖縄県 八重山郡竹富町 高那
9071101 沖縄県 八重山郡竹富町 竹富
9071434 沖縄県 八重山郡竹富町 南風見
9071433 沖縄県 八重山郡竹富町 南風見仲
9071751 沖縄県 八重山郡竹富町 波照間
9071544 沖縄県 八重山郡竹富町 鳩間
9071800 沖縄県 八重山郡与那国町 以下に掲載がない場合
9071801 沖縄県 八重山郡与那国町 与那国
```

このファイルを利用して、標準入力 (キーボード) から 7 桁の郵便番号を入力すると、その郵便番号に対応する地域名を線形探索法[1]を用いて検索し、その結果を表示するプログラム prog11-1.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になります。

prog11-1.c の実行例

```
s1542h017% ./prog11-1
/home/sample/mprog2/zipcode から 121667 件のデータを読み込みました
郵便番号を入力してください : 5202144
5202144 滋賀県 大津市 大萱
郵便番号を入力してください : 5220038
5220038 滋賀県 彦根市 西沼波町
郵便番号を入力してください : 1006110
1006110 東京都 千代田区 永田町山王パークタワー(10階)
郵便番号を入力してください : 9071751
```

[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 135 ページからを適宜参照してください。

```
9071751 沖縄県 八重山郡竹富町 波照間
郵便番号を入力してください : 1234567
対応する地域がありません。
郵便番号を入力してください : .
検索を終了します。
s1542h017%
```

実現の方針 (1) ファイル /home/sample/mprog2/zipinfo から読み取ったデータは、次のような構造体 (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。

```
#define MAX_DATA        (200000)
#define PREF_NAME_LEN   (20)
#define CITY_NAME_LEN   (40)
#define AREA_NAME_LEN   (100)

typedef struct {
    int code;                   /* 郵便番号 */
    char pref[PREF_NAME_LEN];   /* 都道府県名 */
    char city[CITY_NAME_LEN];   /* 市町村名 */
    char area[AREA_NAME_LEN];   /* 町域名 */
} ZipInfo;
```

(2) 配列 data は大きなメモリ領域を必要とするので、関数 main 中では、static を付けて定義 (宣言) する必要がある。

(3) ファイルの入出力には fopen()、fscanf(), fprintf()、fclose() などの標準入出力ライブラリの関数群を使えばよい[2]。例えば、関数 main の定義の冒頭部分で、次のようにしてファイルの内容のデータを読み込むことができる。

```
#define FILE_NAME       "/home/sample/mprog2/zipcode"

int main()
{
    static ZipInfo data[MAX_DATA];
    int num;
    FILE *fp;

    /* 郵便番号の表が格納されたファイルを開く */
    fp = fopen(FILE_NAME, "r");
    /* fopen に失敗したら、エラーメッセージを表示して終了 */
    if (fp == NULL) {
        printf("%s というファイルが読めません\n", FILE_NAME);
        exit(EXIT_FAILURE);
    }

    /* ファイルから配列 data に郵便番号のデータを読み込む */
    num = 0;
    while (fscanf(fp, "%d %s %s %s", &data[num].code,
                  data[num].pref, data[num].city,
                  data[num].area) == 4) {
        num++;
    }
```

[2]ファイルの入出力に関する詳細は C の参考書等を参照して下さい。

```
/* ファイルを閉じる */
fclose(fp);
printf("%s から %d 件のデータを読み込みました\n",
       FILE_NAME, num);
⋮
```

標準入出力ライブラリ関数 fopen を呼び出すことで、ファイルを**開く** (ファイルの読み書きの準備をする) ことができる。第 1 引数はファイルのパス名 (char * 型) であり、第 2 引数は、読み書きの別を表す文字列である (この例中の "r" は読み込みを行うことを表している)。関数 fopen は、読み書きの準備が完了した状態にあるファイルの識別情報を返す。指定されたファイルが存在しないときなど、何らかの障害があってファイルを開くことができない場合は fopen は 0 (NULL) を返す。

(4) 関数 fopen が返すファイルの識別情報は、インクルードファイル stdio.h の中で定義されている FILE というデータ型へのポインタ型であるが、プログラマは FILE の具体的な定義を気にする必要はなく、fscanf などの関数を呼び出してファイルの読み書きを行う際に、この識別情報を引数として渡してやるだけでよい。例えば、fscanf の第 1 引数として、この識別情報を与えると、fscanf は、scanf が標準入力 (通常はキーボード) から読み取って行うのと同じ作業を、fopen で開いておいたファイルから読み取って行ってくれる。

(5) 上のプログラム例の中で、fscanf の書式指定文字列の中に使った「%s」という変換仕様は、空白類 (スペースやタブ、改行文字など) を区切りとして文字列を読み取って、対応する引数 (char * 型) で指定されたアドレスを先頭に格納するためのものである。文字列の終りには (終端を表わす) 0 が置かれる。このプログラムの場合、例えば、fscanf がファイルから

    0640941 北海道 札幌市中央区 旭ケ丘

という行を読み込んだ場合、data[num].code には整数値 640941 が格納され、3 つの文字配列 data[num].pref、data[num].city、data[num].area には、それぞれ "北海道 "、"札幌市中央区 "、"旭ケ丘 " という文字列が格納される。変換仕様 %s を使用する際には、読み込んだ文字列を格納する先となる配列は十分な大きさを持っていなければならないことに注意する必要がある。

(6) 開かれたファイルに対する入出力作業がもう必要なくなったら、関数 fclose を呼んで、そのファイルを**閉じる**。一旦閉じたファイルに対して再度入出力を行いたい場合は、再度 fopen を呼ばなければならないことに注意する。

(7) ファイル中の内容をすべて配列 data に読み込みんだら、関数 main は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、配列 data に対して線形探索を行って、見つかった情報を標準出力 (端末ウィンドウ) に出力すればよい。このとき、実際に線形探索を行う部分は次のように宣言することのできる関数 linearsearch として定義しておき、この関数を main から呼び出

すようにする。

```
int linearsearch(ZipInfo data[], int num, int code);
```

関数 linearsearch は、第 3 引数 code で指定された郵便番号を持つ要素を、配列 data (要素数は num) の先頭から末尾に向って線形探索法で探し、見つかった要素の添字を返す。見つからなかった場合は -1 を返す。関数 main は scanf が整数 (郵便番号) の読み込みに失敗するまで、関数 linearseach の呼び出しと、その結果の表示を繰り返す。

3 演習問題

次の演習問題に取り組み、prog11-1.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。

3.1 prog11-1.c

「2. 線形探索法を用いて郵便番号から住所を検索する」の節のプログラム prog11-1.c を完成しなさい。このプログラムの提出時には教員か TA のチェックが必要となります。

3.2 prog11-2.c

郵便番号表が格納されたファイル /home/sample/mprog2/zipcode をよく調べてみると、

```
5220038 滋賀県 彦根市 西沼波町
5220038 滋賀県 彦根市 山之脇町
```

のように、同じ郵便番号が異る町域に跨って割り当てられていることに気が付きます。prog11-1.c では、最初に見つかった方が表示されるわけですが、余裕のある受講者は、次のような実現の方針に従って、入力された郵便番号を持つものすべてが表示されるようなプログラム prog11-2.c を作って見ましょう。この課題は mprog2 コマンドで提出する必要はありません。

実現の方針 (1) 関数 linearsearch に引数を 1 つ追加して、次のように宣言される関数に変更する。

```
ZipInfo *linearsearch(ZipInfo data[], int start, int num, int code);
```

(2) 変更後の関数 linearsearch は、添字 0 から末尾に向けて配列 data の内容を線形探索するのではなく、(配列の先頭から start 個の要素を無視して) 添字 start を起点に、そこから末尾に向けて探索を行う。

(3) この範囲で見つかった場合に、見つかった要素の添字を返り値として返し、見つからなかった場合は -1 を返す。

(4) 関数 main では、関数 linearsearch を呼んで、その返り値を調べ、見つかった場合は、見つかった要素を出力するとともに、その次の添字を第 2 引数 (start) として、 さらに linearsearch を呼び出し、その返り値が -1 となるまで、繰り返し見つかった要素の出力を行うようにする。

prog11-2.c の実行例

```
s1542h017% ./prog11-2
/home/sample/mprog2/zipcode から 121667 件のデータを読み込みました
郵便番号を入力してください : 5202144
5202144 滋賀県 大津市 大萱
郵便番号を入力してください : 5220038
5220038 滋賀県 彦根市 西沼波町
5220038 滋賀県 彦根市 山之脇町
郵便番号を入力してください : 7794404
7794404 徳島県 美馬郡半田町 青野
7794404 徳島県 美馬郡半田町 折坂
7794404 徳島県 美馬郡半田町 小井野
7794404 徳島県 美馬郡半田町 高清
7794404 徳島県 美馬郡半田町 坂根
7794404 徳島県 美馬郡半田町 下尾尻
7794404 徳島県 美馬郡半田町 下竹
7794404 徳島県 美馬郡半田町 上蓮
7794404 徳島県 美馬郡半田町 日谷尾
郵便番号を入力してください : 1234567
対応する地域がありません。
郵便番号を入力してください : .
検索を終了します。
s1542h017%
```

プログラミングおよび実習Ⅱ・第 11 回・終り

付録. 前回の演習問題の解答例

prog10-1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "util.h"

typedef int Boolean;
#define TRUE            (1)
#define FALSE           (0)

#define NAME_LEN        (100)
#define MAX_DATA        (100000)

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

void selectsort(Record *d[], int num)
{
    Record *tmp;
    int start, pos, i;
    int min;

    for (start = 0; start < num-1; start++) {
        min = d[start]->score;
        pos = start;
        for (i = start+1; i < num; i++) {
            if (d[i]->score < min) {
                min = d[i]->score;
                pos = i;
            }
        }
        if (pos != start) {
            tmp = d[start];
            d[start] = d[pos];
            d[pos] = tmp;
        }
    }
}

typedef int Mode;
#define RANDOM  (1)
#define INC     (2)
#define DEC     (3)

void initdata(Record d[], int num, Mode mode)
{
    int i;

    switch (mode) {
    case RANDOM:
        for (i = 0; i < num; i++)
            d[i].score = rand();
        break;
    case INC:
        for (i = 0; i < num; i++)
            d[i].score = i;
```

```
        break;
    case DEC:
        for (i = 0; i < num; i++)
            d[i].score = num - i;
        break;
    }
}
#define MIN_SEC         (10.0)
#define INIT_SEC        (3.0)

typedef struct {
    int repeat;         /* 整列を繰り返した回数 */
    double t1;          /* データの初期化に要した時間(秒) */
    double t2;          /* データの初期化と整列に要した時間(秒) */
} ExpResult;

ExpResult checktime(int num, double sec, Mode mode)
{
    static Record data[MAX_DATA], *dp[MAX_DATA];
    ExpResult r;
    int i, j;

    /* 名前の部分を初期化 */
    for (i = 0; i < num; i++)
        strcpy(data[i].name, "Test");

    /* 先にデータ生成と整列に必要な時間を求める */
    clock_reset();
    clock_on();
    r.repeat = 0;
    do {
        initdata(data, num, mode);
        for (i = 0; i < num; i++)
            dp[i] = &data[i];
        selectsort(dp, num);
        r.repeat++;
    } while (clock_time() < sec);
    r.t2 = clock_off();

    /* データ生成や clock_time の呼び出しのためだけに必要な時間を求める */
    clock_reset();
    clock_on();
    for (j = 0; j < r.repeat; j++) {
        initdata(data, num, mode);
        clock_time();
    }
    r.t1 = clock_off();
    return r;
}

int main(int argc, char *argv[])
{
    int num;
    Mode mode;
    ExpResult r;

    if (argc != 3) {
        printf("要素数と初期化の方法が指定されていません。\n");
        exit(EXIT_FAILURE);
    }
```

```
    if (sscanf(argv[1], "%d", &num) != 1 || num <= 0 || num > MAX_DATA) {
        printf("要素数の指定が正しくありません。\n");
        exit(EXIT_FAILURE);
    }

    if (strcmp(argv[2], "random") == 0) {
        mode = RANDOM;
    }
    else if (strcmp(argv[2], "inc") == 0) {
        mode = INC;
    }
    else if (strcmp(argv[2], "dec") == 0) {
        mode = DEC;
    }
    else {
        printf("初期化方法の指定が正しくありません。\n");
        exit(EXIT_FAILURE);
    }

    /* 試験的に INIT_SEC 秒間計測 */
    r = checktime(num, INIT_SEC, mode);
    /* 純粋な整列時間の累計が MIN_SEC 以上になるように計測 */
    r = checktime(num, MIN_SEC*r.t2/(r.t2-r.t1), mode);
    /* 結果を出力する */
    printf("%d, %.3f, %.3f, %d, %.10f\n",
           num, r.t2, r.t1, r.repeat, (r.t2 - r.t1) / r.repeat);
    return EXIT_SUCCESS;
}
```