目次

★7　探索

この授業の「前半（高橋担当分）の後半」では，「郵便番号簿の探索」をするプログラムの作成を目指す．

★7.1　探索とは

探索（または**検索**）とは，データ集合の中からデータを探し出す処理のことである．例えば，電話番号と氏名を組にした表の中から，1234567 という電話番号に一致するものを探し出す（一致するものが存在しなければそう知らせる），というようなことを実現するために必要となる．この例の 1234567 のような，探索の条件となる値のことを**キー**と言う．

郵便番号データの例

```
6148062 京都府八幡市八幡清水井
3500263 埼玉県坂戸市堀込
9518142 新潟県新潟市関屋大川前
9401103 新潟県長岡市曲新町
5200533 滋賀県滋賀郡志賀町小野朝日
5190438 三重県度会郡玉城町原
5220353 滋賀県犬上郡多賀町月之木
1057219 東京都港区東新橋汐留メディアタワー (19 階)
   :              :
```

探索プログラムの動作例

```
121667 件の郵便番号データを読み込みました
7 桁の郵便番号を入力して下さい（0 で終了） 5202123
[20409] 5202123 滋賀県大津市瀬田大江町
7 桁の郵便番号を入力して下さい（0 で終了） 0010010
[49628] 0010010 北海道札幌市北区北十条西 (1〜4 丁目)
7 桁の郵便番号を入力して下さい（0 で終了） 9998525
[36801] 9998525 山形県飽海郡遊佐町直世
7 桁の郵便番号を入力して下さい（0 で終了） 1234567
見つかりませんでした
7 桁の郵便番号を入力して下さい（0 で終了） 0
```

探索のやり方としては，「キーの値を区間で指定する」(☆ 1) などの場合も考えられるが，この授業では単純に「**キーと一致するデータを探す**」場合のみを考える．また，簡単のため**キーと一致するデータは二つ以上は存在しない**（もしくは一致するデータが二つ以上複数あっても一つ見つけたら探索を打ち切る）ものとする．

☆ 1) 例えばキー値が 25 以上 35 未満など

探索の手法は，データの格納にどのようなデータ構造を用いるかによって大まかに二つに分類できる (☆ 2)．

- **単純に配列を用いるもの** 線形探索，二分探索
- **データ構造を工夫するもの** 二分探索木や連結リストを用いる方法

☆ 2) 二分探索やハッシュ法といったアルゴリズムについてや，二分探索木や連結リストといったデータ構造については，この授業では解説しない．興味のある人は関係の授業や書籍等で学んでね．

また，データ構造が単純な配列であるか否かとは独立に，データの格納の仕方を工夫して探索の効率を上げる手法として，**ハッシュ法**と呼ばれるものもある．実際にデータ探索を行うプログラムを作る際には，データの追加・削除の有無，とにかく探索が速ければよいのか他の処理もできる必要があるのか，などを考慮してデータ構造や手法を選ぶことになる．しかし，この授業ではその辺りのことは省略し，自分でプログラムを作成してみるのは，最も単純な探索手法である線形探索のみとする．

★ 7.2 線形探索

線形探索 (☆ 3) は，データを順番にチェックしていくだけの，最も単純な探索手法である．前述の二分探索やハッシュ法といったアルゴリズムと比べれば探索にかかる計算コストが大きい（ので時間がかかる）けれど，非常に単純なプログラムで実現することができる．

例えば，次のように `int` 型の配列 `x` の先頭から順に `n = 6` 個のデータが格納されている場合を考えてみよう．このとき，線形探索では，単純に配列の先頭から順にキーに一致するデータがないか探していく．

(1) キー 901334 を探索　　(2) キー 901234 を探索

i	x[i]
0	901051
1	901002
2	901359
3	901334
4	900555
5	901000
6	
:	

線形探索では，データ数 n に対して，探索成功 (キーと一致する値が配列中にあった) の場合，キーと配列の要素との比較を平均 $n/2$ 回（最小 1 回，最大 n 回）行うことになる．一方，探索失敗（キーと一致する値がない）の場合，キーと一致する要素が配列中に存在しないことを確定させるためには配列中の全ての要素との比較（すなわち n 回の比較）が必要となる．したがって，探索に必要な比較回数は，成功の場合も失敗の場合も $O(n)$ ということになる (☆ 4).

☆ 3) 線形探索: linear search, sequential search, **順探索**, **逐次探索**とも言う．

☆ 4) ［発展］アルゴリズムやデータ構造について学ぶとわかるかもしれない話．データ数 n のときの線形探索の時間計算量が $O(n)$ であるのに対して，二分探索のそれは $O(\log n)$ である．また，ハッシュ法は，「探索に関する時間計算量がデータ数によらない」という興味深い性質をもっている．ただし，いずれの手法にも線形探索と比較して劣っている所もあるので, 実用的に線形探索を用いることもある．

★8 構造体

★8.1 構造体を使おう

とある健康診断で得られたデータを扱うプログラムを作ることを考える．データは，「名前」「身長」「体重」の 3 つが組になっているとする．kenshin1.c は，2 人分のデータ（○○ A という変数が 1 人目の分，○○ B が 2 人目の分を表す）のキー入力を受け取り，それらを交換してから表示しようとしたものである．

このようにいくつかの変数が組になっているような場合には，それらをひとまとめにした**構造体**を定義して使うのがよい．実際，kenshin1.c には誤りが含まれているのにそれに気がつきにくいが，kenshin2.c のように構造体を使うとソースを見通しよく書けてバグも混入しにくくなる．

● 構造体の作り方・使い方

(1)

(2)

(3)

kenshin1.c

```
 1  #include <stdio.h>
 2  #include <string.h>  // for strcpy()
 3
 4  int main(void)
 5  {
 6    char nameA[100], nameB[100];
 7    double heightA, heightB;
 8    double weightA, weightB;
 9    char ntmp[100];
10    double htmp, wtmp;
11
12    scanf("%s %lf %lf", nameA, &heightA, &weightA);
13    scanf("%s %lf %lf", nameB, &heightB, &weightB);
14
15    // A <--> B
16    strcpy(ntmp, nameA);
17    htmp = heightA;  wtmp = weightA;
18    strcpy(nameA, nameB);
19    heightA = heightB;  weightA = weightB;
20    strcpy(nameB, ntmp);
21    heightB = htmp;
22
23    printf("[A] %s %3.1fcm %3.1fkg\n",
24           nameA, heightA, weightA);
25    printf("[B] %s %3.1fcm %3.1fkg\n",
26           nameB, heightB, weightB);
27    return 0;
28  }
```

実行例

```
$ ./kenshin1
Hogeo 169.8 65.7      ← キー入力
Fugayo 203.4 45       ← キー入力
[A] Fugayo 203.4cm 45.0kg
[B] Hogeo 169.8cm 45.0kg
```

kenshin2.c

```
 1  #include <stdio.h>
 2
 3
 4
 5
 6
 7
 8
 9  int main(void)
10  {
11
12
13    scanf("%s %lf %lf", a.name, &a.height, &a.weight);
14    scanf("%s %lf %lf", b.name, &b.height, &b.weight);
15
16    tmp = a; a = b; b = tmp;  // A <=> B
17
18    printf("[A] %s %3.1fcm %3.1fkg\n", a.name, a.height, a.weight);
19    printf("[B] %s %3.1fcm %3.1fkg\n", b.name, b.height, b.weight);
20    return 0;
21  }
```

★8.2 typedef による型定義

typedef を用いると，自分で新しい型 (type) を定義する (define) ことができる．例えば，右の場合，Kazu という型が新しく定義されている．このプログラム中では，整数を格納する変数の型名として int と同じ意味で Kazu を使える．また，typedef の所を

```
typedef double Kazu;
```

とすれば，今度は Kazu 型は double 型と同じになる．このように，typedef はわかりやすく保守しやすいプログラムの作成に役立つ．

typedef は，構造体の型名を短く表すために利用されることが多い．例えば，上記の struct kcard という構造体の定義の後で右の例 (2) のように typedef しておけば，struct kcard と書くかわりに KCard と書いて済ませることができるようになる（struct kcard と書いてもよい)．

また，typedef による構造体型名の定義は，右の例 (3),(4) のように構造体の定義と組み合わせてしまうこともできる．この場合，例 (4) では構造体タグ名 (例 (2),(3) の kcard) を省略しているので，struct なんたらとは書けず，KCard という形名のみが使えることになる．

typedef の使用例 (1)

```
typedef int Kazu;

Kazu Add(Kazu x, Kazu y){
   return x + y;
}
```

typedef の使用例 (2)

```
struct kcard{
   char name[100];
   double height;
   double weight;
};
typedef struct kcard KCard;

KCard a, b, tmp;
```

typedef の使用例 (3)

```
typedef struct kcard{
   char name[100];
   double height;
   double weight;
} KCard;

KCard a, b, tmp;
```

typedef の使用例 (4)

```
typedef struct {
   char name[100];
   double height;
   double weight;
} KCard;

KCard a, b, tmp;
```

★8.3 構造体と関数

普通の変数と同様に，構造体変数も関数の引数とできる (☆ 5)．以下のプログラムとその実行例 (☆ 6) からわかるように，関数 Futorou を呼び出しても main の側の b.weight の値は変化しないが，関数 Futorou2 のようにポインタを渡せば，関数内で構造体メンバの値を変化させることができる．これは，普通の変数の場合と同じである．ただし，構造体変数へのポインタを用いる場合，プログラムの書き方に注意が必要である．kcard.c の 19 行目に示すように，構造体変数へのポインタを用いてメンバを参照する際には"->"という演算子を用いる．x->weight は，(*x).weight を略記したものである．

☆ 5) 説明は省略するが，関数の戻り値に構造体を指定することもできる．

☆ 6) BMI(Body Mass Index) は，身長 (単位は [m]) と体重 ([kg]) から計算される人間の肥満度指数．22 が標準らしい． BMI = (体重)/(身長)^2

kcard.h

```
 1  #ifndef _KCARD_H
 2  #define _KCARD_H
 3
 4  // 構造体の定義
 5  typedef struct {
 6    char name[100];
 7    double height;
 8    double weight;
 9  } KCard;
10
11  // プロトタイプ宣言
12  double BMI(KCard x);
13  void Pocha(KCard x);
14  void Pocha2(KCard *x);
15
16  #endif
```

kcard.c

```
 1  #include "kcard.h"
 2
 3  // 引数で構造体変数を受け取る関数 (1)
 4  double BMI(KCard x)
 5  {
 6    double h = x.height / 100;
 7    return x.weight / (h * h);
 8  }
 9
10  // 引数で構造体変数を受け取る関数 (2)
11  void Pocha(KCard x)
12  {
13    x.weight += 30;
14  }
15
16  // 引数で構造体変数を受け取る関数 (3)
17  void Pocha2(KCard *x)
18  {
19    x->weight += 30;
20  }
```

kenshin3.c

```
 1  #include <stdio.h>
 2  #include "kcard.h"
 3
 4  int main(void)
 5  {
 6    KCard a, b, tmp;
 7
 8    scanf("%s %lf %lf", a.name, &a.height, &a.weight);
 9    scanf("%s %lf %lf", b.name, &b.height, &b.weight);
10
11    printf("%s さんの体重は %f[kg], BMI は %f です\n", a.name, a.weight, BMI(a));
12    printf("%s さんの体重は %f[kg], BMI は %f です\n", b.name, b.weight, BMI(b));
13    printf(" Pocha  => "); Pocha(b);
14    printf("%s さんの体重は %f[kg], BMI は %f です\n", b.name, b.weight, BMI(b));
15    printf(" Pocha2 => "); Pocha2(&b);
16    printf("%s さんの体重は %f[kg], BMI は %f です\n", b.name, b.weight, BMI(b));
17    return 0;
18  }
```

実行例

```
$ ./kenshin3
Hogeo 169.8 65.7
Fugayo 203.4 45
Hogeo さんの体重は 65.700000[kg], BMI は 22.787149 です
Fugayo さんの体重は 45.000000[kg], BMI は 10.877037 です
 Pocha  => Fugayo さんの体重は 45.000000[kg], BMI は 10.877037 です
 Pocha2 => Fugayo さんの体重は 75.000000[kg], BMI は 18.128395 です
```

補足: scanf の戻り値

いつもお世話になっている関数 scanf は，実は整数値を返すように定義されている (☆7)．この戻り値は，
「scanf の引数に指定された書式に従って入力を解釈してみた結果，

」
を表す．例えば，

```
int x, y, rv;
rv = scanf("%d %d", &x, &y);
```

というプログラム (の一部) 実行した場合，「123 -5」というキー入力を与えたとすると，変数 rv の値は 2 になる．一方，「123 abc」だと rv は 1 になる．また，「hoge -5」だと 0 になる．前者の場合，変数 x の方には正しく読み込めた値 (123) が代入される．この性質を利用すると，次のように，入力データの件数をプログラム自身に数えさせることができる．このプログラムでは，浮動小数点数と整数の組を受け取れなかった時点または配列がいっぱいになった時点で入力の繰り返しを終了するようになっている．

☆ 7) fscanf 等も同様

☆ 8) ファイルの終わりに達したなどの理由で入力エラーが発生した場合，EOF が返される (EOF は stdio.h で定義された定数)．

scanftest.c

```
 1  #include <stdio.h>
 2
 3  #define N 100
 4
 5  int main(void)
 6  {
 7    double x[N];
 8    int y[N], i, n, rv;
 9
10    for(n = 0; n < N; n++){
11      rv = scanf("%lf %d", &x[n], &y[n]);
12      if(rv != 2) break;
13    }
14    printf("%d 件読み込んだで\n", n);
15    for(i = 0; i < n; i++){
16      printf("[%d] %f %d\n", i, x[i], y[i]);
17    }
18    return 0;
19  }
```

実行例

```
$ ./scanftest
1.23 456
7 890
0.1
   1026
3.14  hoge
3 件読み込んだで
[ 0] 1.230000 456
[ 1] 7.000000 890
[ 2] 0.100000 1026
```