アルゴリズムとデータ構造 第 2 回課題 解答

問題：以下のプログラムは，ユーザから入力された 100 個の点の座標について最も近い 2 点の間の距離を表示するものである．

(1) 空欄（薄く塗られた部分）では何を行うべきか考え，行うべき処理を C 言語で記せ．なお，新たに変数を使う場合は変数の宣言を追加せよ．

(2) このプログラムの効率性を評価したい．関数 dist()が呼び出される回数ができるだけ少ない方が効率がよいものと考える．自分の考えたやり方では，呼出回数は何回になるかを理由をつけて示せ（裏面を利用せよ）

考え方： NUM 個の座標データが 2 次元配列 p[NUM][2]の中に入れられており，2 点間の距離の中で最短のものを探すという問題である．そこで，NUM 個の座標の中から 2 個の座標をとる全ての組み合わせについて距離を計算し，その中で最も小さかったものが答えとなる．

組み合わせの数は ${}_{NUM}C_2$ 通り．これを網羅的に計算するためには，**二重のループ構造**を用いればよい．1 点目を選択するループでは，0 番の点から順番に NUM-2 番（最後の一つ手前）までを選択する．1 点目に選んだ番号を i とすれば，2 点目に選ぶべき番号 j は i+1 番目から NUM-1 番目（最後）までであることが分かる（1 番目に選んだ座標と同じ座標や 1 番目に選んだものより番号の若い座標を選択してはならない理由は分かるだろうか？）．

このようにして二重ループを使えば，全ての 2 点の組み合わせに対して距離計算を行うことはできる．では，その中で最短の距離を求めるにはどうすればよいだろうか？

一般的には，**「今のところまでの最小値」を一つの変数として保持し，現在までの最小値より更に小さな値が見つかったらその変数を更新するという方法**がとられる．「今のところの最小値」としての変数を min とする．まずは，min を初期化する．「あり得ないぐらい大きな値（解答例では 1.0e100＝1.0×10^{100} とした）」で初期化する方法と，一つのサンプル（例えば dist(p[0], p[1])）を使う方法がある．次に，先程の方法で，全ての 2 点の組み合わせに対して距離を計算し，min よりも距離が小さいときは，min をその距離に更新する．こうして全ての組み合わせに対して距離計算が終了した時点で，変数 min には全体における最小値が入っている．

実装においては，最初に変数 min を初期化し，次に 2 重の for 文を使って 2 重ループを実現する．

繰り返し回数については，2 点間の距離を計算するつど関数 dist()が一回呼び出されるとすれば，その回数は NUM 個から 2 個を取り出す組み合わせの数に等しく，${}_{NUM}C_2$ である．

プログラムの解説： 解答例に示したコードに即して解説を行う．01～03 行はプリプロセッサ指示であり，01, 02 行ではヘッダファイル stdio.h および math.h を取り込んでおり，03 行では文字列 NUM を 100 と読み替える置換を指示している．

05 行は関数プロトタイプ宣言であり，「このプログラムでは，double 型の返値をとり，double 型へのポインタの 2 つの引数を持つ dist という名前の関数を用いる」と宣言している．

07 行は main 関数の宣言で，引数・返り値ともに持たないことを示している（通常は main 関数は返り値を持つ．この返り値はシステム（プログラムを読みだしたプログラムで，Windows では通常，コマンドプロンプトあるいはエクスプローラである）に対して返されるものである．

09～11 行が main 関数内でのローカル変数の宣言であり，main 関数の中ではこれこれこのような型・名前の変数を使うよ，と宣言している．ここで注意して欲しかったのは，なぜか j という変数が定義されていること．「二重ループを使うよ」というヒントになっているつもりで入れた．さらに，min がすでに定義されているのもヒントである．

13, 14 行目では，ユーザから scanf を使って座標値の x 座標，y 座標を入力してもらい，これを配列の p[i][0]，p[i][1]に記録する．

21 行は，求まった min を printf で画面に出力している部分．double 型については，scanf では%lf，printf では%f を使う．

24 行目以降は，2 つの座標値（それぞれ要素数 2 の 1 次元配列）の間の距離を求めて返す関数．x 座標の差の二乗と y 座標の差の二乗を足して平方根（square root）をとったものを返している．

解答：

(1) 解答コード例は以下の通り：

```
01: #include <stdio.h>
02: #include <math.h>
03: #define NUM 100
04:
05: double dist(double *, double *);
06:
07: void main(void)
08: {
09:    int    i, j;
10:    double  p[NUM][2];
11:    double  min = 1.0e100;
12:
13:    for (i = 0; i < NUM; i++)
14:       scanf("%lf %lf", &p[i][0], &p[i][1]);
15:
16:    for (i = 0; i < NUM; i++)
17:       for (j = i+1; j < NUM; j++)
18:         if (min > dist(p[i], p[j]))
19:           min = dist(p[i], p[j]);
20:
21:    printf("Shortest distance = %lf¥n", min);
22: }
23:
24: double dist(double *p0, double *p1)
25: {
26:    return(sqrt((p0[0] - p1[0]) * (p0[0] - p1[0]) +
27:                (p0[1] - p1[1]) * (p0[1] - p1[1])));
28: }
```

ただし，このコードでは 2 点間の距離を min と比較するために dist()を呼び出し，さらに min を更新する場合は再び同じ引数で dist()を呼び出している（下線部）ので，効率が悪い．これを回避するには，double 型の変数（例えば d）を作っておいて，以下のようにすればよい．

```
16:       min = 1.0e100;
17:       for (i = 0; i < NUM-1; i++)
18:            for (j = i+1; j < NUM; j++) {
19:                 d = dist(p[i], p[j]);
20:                 if (min > d) min = d;
21:            }
```

(2) 上記の改良したコード（変数 d を使うバージョン）では，100 個の座標から 2 個の座標を取り出す全ての組み合わせに対して 1 回ずつ距離計算を行うので，${}_{100}C_2=4950$ 回．改良前のコードでは，比較のための呼出に加えて min を更新するたびに関数 dist()を余計に呼び出してしまう．その更新回数は最悪 4950 回(毎回)，最良で 0 回（初期値が最小の場合）．つまりこの場合は回数を正確に特定できず，4950 回～9900 回となる．なお，min の初期化を dist(p[0], p[1])により行った場合はさらにその 1 回が加わる．

より一般的に，座標の数が n 個であった場合は，繰り返し回数は ${}_nC_2+1=n(n\ \ 1)/2+1$ 回となる．