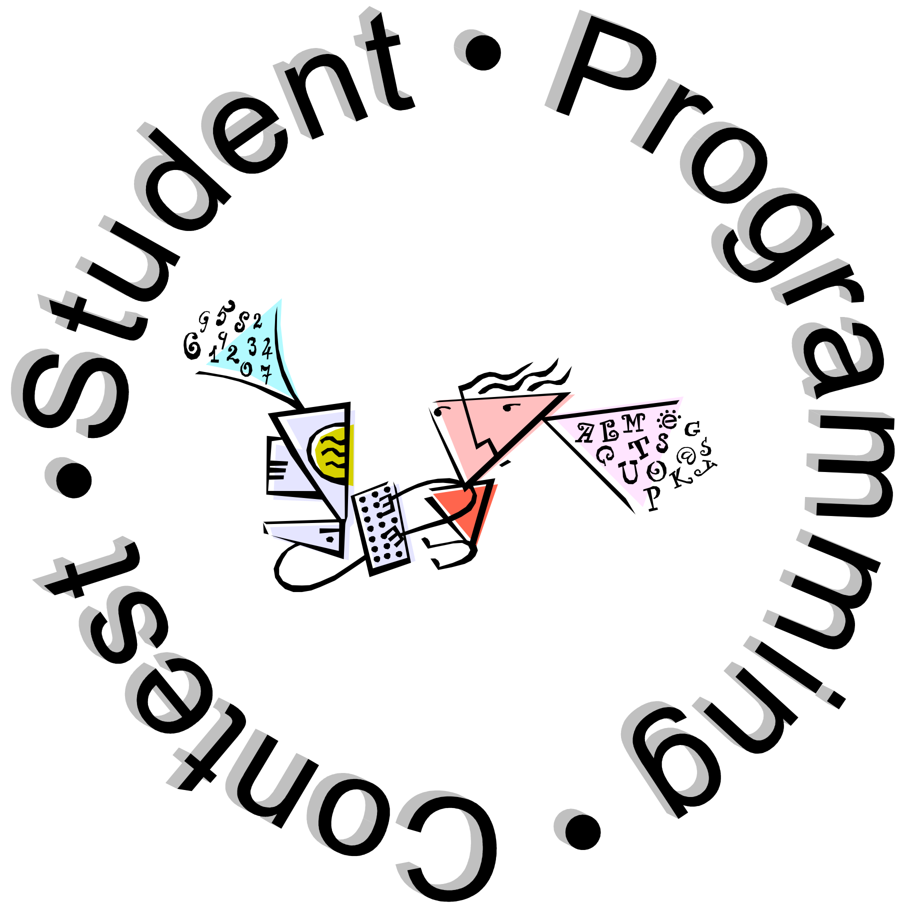
The Eighth Annual

Student Programming Contest

of the
CCSC Southeastern Region



Saturday, November 3, 2001
8:00 A.M. – 12:00 P.M.

What the Hail

The Problem

There is an interesting series of integers known as *hailstones*. Hailstones are formed by being given a starting integer and generating the next integer based on the one that immediately precedes it in the series as follows: If the previous integer was even, the next integer in the series is half of it. If the previous integer was odd, the next integer is three times it plus one. Although the series goes up and down (like hailstones before they fall to the ground), it eventually settles into a steady state of 4, 2, 1, 4, 2, 1, … For example, starting at 21, the hailstone series is: 21, 64, 32, 16, 8, 4, 2, 1, 4, 2, 1, … For 21, the series required five steps before the steady state was reached.

Write a program that computes the number of steps necessary to reach the steady state in the hailstone series beginning at a given positive integer.

Sample Input

Your program must take its input from the ASCII text file `prob1.in`. The file contains a sequence of one or more positive integer values, one per line. Sample contents of the file could appear as:

```
21
10
```

Sample Output

Your program must direct its output to the screen and must tell the number of steps required to reach the hailstone steady state for each integer recorded in the input file. Your program must format its output exactly as that shown below which is the output corresponding to the sample input above.

```
5 steps were necessary for 21.
4 steps were necessary for 10.
```

Wire Routing

The Problem

A common approach to the write-routing problem for electrical circuits is to impose a grid over the wire-routing region. The grid divides the routing region into an $n \times m$ array of squares, much like a maze. A wire runs from the midpoint of one square *a* to the midpoint of another square *b*. In doing so, the wire may make right-angle turns. Grid squares that already have a wire or some other obstruction through them are blocked and cannot be used. To minimize signal delay, we wish to route the wire using a shortest path between *a* and *b*. Figure 2a shows a 7×7 circuit board grid. The gray squares are blocked and the white squares are clear. The square labeled *a* is the starting point of a wire path we wish to construct and the square labeled *b* is the end point of this path. Figure 2b shows the same grid with a shortest wire path traced with a line. Notice that a shortest wire path is not necessarily unique.

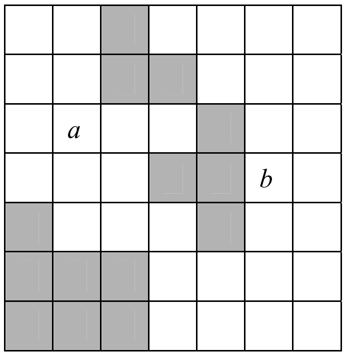


Figure 2a.

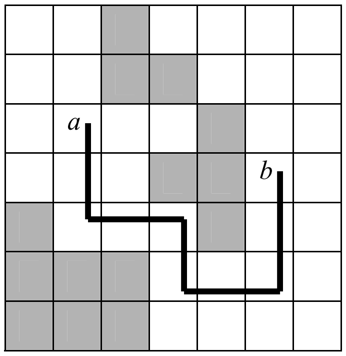


Figure 2b.

Write a program that computes the length of the shortest wire path given a circuit board grid, a starting point, and an ending point.

Sample Input

Your program must take its input from the ASCII text file `prob2.in`. The file contains a sequence of one or more wire path specifications: The first line contains the grid size *n* ($n \times n$ grid), the second line contains a (row, column) ordered pair representing the starting point of the desired wire path, the second line contains a (row, column) ordered pair representing the ending point of the desired wire path, and the following *n* lines contains a row-major specification of the grid. Each of these *n* lines contains *n* entries separated by exactly one blank space. Each entry is either a 0 (open square) or a 1 (blocked square). Thus, an entire wire path specification spans exactly *n*+3 lines. You are guaranteed that the data contains no errors.

Contents of the file for Figure 2a would appear as:

```
7
3 2
4 6
0 0 1 0 0 0 0
0 0 1 1 0 0 0
0 0 0 0 1 0 0
0 0 0 1 1 0 0
1 0 0 0 1 0 0
1 1 1 0 0 0 0
1 1 1 0 0 0 0
```

Sample Output

Your program must direct its output to the screen and must tell the length (number of squares, including start and end points) of a shortest wire path from the starting square to the ending square, if such a path exists. Your program must format its output exactly as that shown below which is the output corresponding to the sample input above.

```
There are 10 squares on a shortest path from (3,2) to (4,6).
```

Crossover

The Problem

An electrical routing channel has *n* wiring pins both at the top and bottom of the channel and wires are used to connect a pin at the top of the channel to a pin at the bottom of the channel. The 10-pin routing channel and its wiring connections shown in Figure 3 can be specified as C = [8, 7, 4, 2, 5, 1, 9, 3, 10, 6]. The cardinality of C, in this case 10, tells the number of pins at the top and bottom, and the wiring connections are specified in that pin *k* at the top of the channel is connected with a straight-line wire to pin C_k at the bottom of the channel. Thus, in this example pin 1 at the top is connected to pin 8 at the bottom; pin 2 at the top is connected to pin 7 at the bottom; and so on. Notice that there are 22 wire crossings in Figure 3.

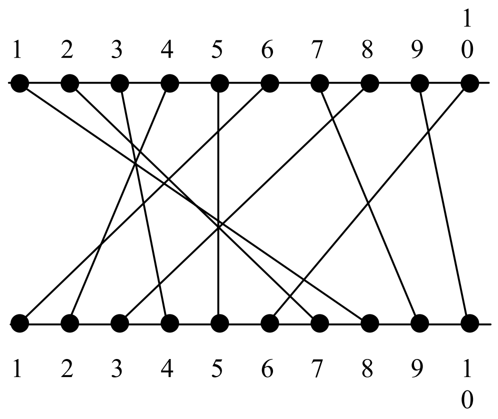


Figure 3.

Write a program that computes the total number of wire crossings in a given routing channel.

Sample Input

Your program must take its input from the ASCII text file `prob3.in`. The file contains a sequence of one or more routing channel specifications, one per line. Each line contains *n* integers separated by exactly one blank space. The specific value of *n* can vary from line to line. Each integer specifies a top pin to bottom pin connection as discussed above. You are guaranteed that the data contains no errors.

Sample contents of the file could appear as that shown below. Notice that the first line specifies the routing channel depicted in Figure 3.

```
8 7 4 2 5 1 9 3 10 6
3 1 4 5 2
```

Sample Output

Your program must direct its output to the screen and must tell the total number of wire crossings for each routing channel specification in the input. Your program must format its output exactly as that shown below which is the output corresponding to the sample input above.

```
There are 22 wire crossings in routing channel 1.
There are 4 wire crossings in routing channel 2.
```

Target Practice

The Problem

You have been tasked to develop a scoring program for an electronic version of darts. The user is presented with a target composed of ten concentric circles, with the inner-most circle carrying the most points if hit. The points associated with a circle decreases as its radius increases. No points are given for a hit outside the target, that is, outside the outer-most circle.

The center of the target lies at the center of a window on the screen. The radius of the inner-most circle is 50 pixels, and the radius of each of the remaining circles is 50 pixels greater than the circle just within it. Thus, the outer-most circle has a radius of 500 pixels. The player of the darts game shoots a Nintendo-like gaming gun at the screen and the game's software records the (x,y) coordinate of the "hit." The game window is viewed by the software as a Cartesian plane with the origin (0,0) at the center of the window. The game player gets five shots and then the software tallies the score. Points are distributed as follows: circle 1 (inner-most circle) = 500 points; circle 2 = 300 points; circle 3 = 250 points; circle 4 = 200 points; circle 5 = 150 points; circle 6 = 100 points; circle 7 = 75 points; circle 8 = 50 points; circle 9 = 25 points; circle 10 (outer-most circle) = 10 points; outside all circles = 0 points. If a hit is on a circle boundary, the hit is considered to be in the smaller circle, not the larger one.

Write a program that performs the scoring function of the electronic darts game.

Sample Input

Your program must take its input from the ASCII text file `prob4.in`. The file consists of one or more game summaries. Each game summary records the game player's name and the (x,y) coordinates of each of their five shots. Thus, each game summary consists of six lines: the first contains the player's name and the remaining five lines record the shot coordinates, one per line, with the x (horizontal) coordinate appearing first and the y (vertical) coordinate appearing second. Exactly one blank space separates the x and y coordinates. Sample contents of the file could appear as:

```
Frank
75 200
0 1
650 780
300 490
15 344
Sally
0 0
95 63
100 250
378 -97
1 1
```

Sample Output

Your program must direct its output to the screen and must report a score summary for each player recorded in the input file. The score summary must indicate the player's name, the points associated with each hit, and the total score for the game. Your program's output must for formatted exactly as shown below, which is the output corresponding to the sample input given above.

```
Score Summary for Frank
-----------------------
   Hit 1 =          150
   Hit 2 =          500
   Hit 3 =            0
   Hit 4 =            0
   Hit 5 =           75
              ---------
   Score =          725


Score Summary for Sally
-----------------------
   Hit 1 =          500
   Hit 2 =          250
   Hit 3 =          100
   Hit 4 =           50
   Hit 5 =          500
              ---------
   Score =         1400
```

Parse the Prefix, Please

The Problem

Three standard ways of representing arithmetic expressions are in prefix, infix, and postfix notation. In prefix, a binary operator immediately precedes its two operands; in infix a binary operator is placed between its two operands; in postfix a binary operator immediately follows its two operands. For example, here are three equivalent forms of the same arithmetic expression:

Prefix: + 3 4
Infix: 3 + 4
Postfix: 3 4 +

Although we are accustomed to using infix notation, it isn't as compact as the other two because of the need for parentheses. For example, to add 5 to 4 then multiply the result by 8 we would have to use parentheses in infix notation to force the addition to be done before the multiplication, while prefix and postfix would not need them:

Prefix: * + 5 4 8
Infix: (5 + 4) * 8
Postfix: 5 4 + 8 *

Write a program which accepts prefix arithmetic expressions involving single-digit operands and the operators + (addition), - (subtraction), * (multiplication), and / (division) and outputs an equivalent infix expression, fully parenthesized.

Sample Input

Your program must take its input from the ASCII text file `prob5.in`. This file consists of an undetermined number of prefix expressions, one per line. There is no whitespace between operators and operands. Sample contents of the file could appear as:

```
+ 3 4
* + 5 4 8
```

Sample Output

Your program must direct its output to the screen and must format the output as shown below. The correct output for the sample input given above is:

```
(3 + 4)
((5 + 4) * 8)
```

The Domino Effect

The Problem

A standard set of Double Six dominoes contains 28 pieces (called bones) each displaying two numbers from 0 (blank) to 6 using dice-like pips. The 28 bones, which are unique, consist of the combination of pips shown in Figure 6a. Figure 6b depicts bone 16 as you would see it in a physical set of dominoes.

Bone #	Pips		Bone #	Pips		Bone #	Pips		Bone #	Pips	
1	0	0	8	1	1	15	2	3	22	3	6
2	0	1	9	1	2	16	2	4	23	4	4
3	0	2	10	1	3	17	2	5	24	4	5
4	0	3	11	1	4	18	2	6	25	4	6
5	0	4	12	1	5	19	3	3	26	5	5
6	0	5	13	1	6	20	3	4	27	5	6
7	0	6	14	2	2	21	3	5	28	6	6

Figure 6a

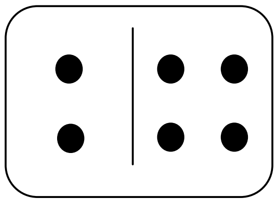


Figure 6b

All the Double Six dominoes in a set can be laid out to display a 7 × 8 grid of pips. Each layout corresponds to at least one "map" of the dominoes. A map consists of an identical 7 ×8 grid with the appropriate bone numbers substituted for the pip numbers appearing on that bone. An example 7 × 8 grid display of pips is shown in Figure 6c, while a corresponding map of bone numbers is shown in Figure 6d.

6	6	2	6	5	2	4	1
1	3	2	0	1	0	3	4
1	3	2	4	6	6	5	4
1	0	4	3	2	1	1	2
5	1	3	6	0	4	5	5
5	5	4	0	2	6	0	3
6	0	5	3	4	2	0	3

Figure 6c

28	28	14	7	17	17	11	11
10	10	14	7	2	2	21	23
8	4	16	25	25	13	21	23
8	4	16	15	15	13	9	9
12	12	22	22	5	5	26	26
27	24	24	3	3	18	1	19
27	6	6	20	20	18	1	19

Figure 6d

Write a program that will analyze the pattern of pips in any 7 × 8 layout of a standard set of dominoes and produce a map showing the position of all dominoes in the set. If more than one arrangement of dominoes yield the same pattern, your program should generate a map of each possible layout.

Sample Input

Your program must take its input from the ASCII text file `prob6.in`. This file consists of an undetermined number of domino grid specifications. Each specification consists of seven lines of eight integers from 0 through 6, representing a pattern of pips. Each specification corresponds to a legitimate configuration of bones (that is, there will be at least one map possible for each specification). There is no intervening data separating specifications. Sample contents of the input file could appear as:

```
6 6 2 6 5 2 4 1
1 3 2 0 1 0 3 4
1 3 2 4 6 6 5 4
1 0 4 3 2 1 1 2
5 1 3 6 0 4 5 5
5 5 4 0 2 6 0 3
6 0 5 3 4 2 0 3
5 4 3 6 5 3 4 6
0 6 0 1 2 3 1 1
3 2 6 5 0 4 2 0
5 3 6 2 3 2 0 6
4 0 4 1 0 0 4 1
5 2 2 4 4 1 6 5
5 5 3 6 1 2 3 1
```

Sample Output

Your program must direct its output to the screen and must format the output as shown below. Correct output consists of a grid specification label (beginning with Grid #1) followed by a printing of the grid specification itself. This is followed by a map label for the set and the map(s) that correspond to the grid specification. (Multiple maps can be output in any order.) Numbers in a grid or map must be aligned vertically. After all maps for a grid specification have been printed, a summary line stating the number of possible maps appears. At least three lines are skipped between the output from different grid specifications while at least one line separates the labels, grid printing, and maps within the same problem set. The output corresponding to the sample input given above is:

```
Grid #1:

6  6  2  6  5  2  4  1
1  3  2  0  1  0  3  4
1  3  2  4  6  6  5  4
1  0  4  3  2  1  1  2
5  1  3  6  0  4  5  5
5  5  4  0  2  6  0  3
6  0  5  3  4  2  0  3

Maps resulting from grid #1 are:

28 28 14 7  17 17 11 11
10 10 14 7  2  2  21 23
8  4  16 25 25 13 21 23
8  4  16 15 15 13 9  9
12 12 22 22 5  5  26 26
27 24 24 3  3  18 1  19
27 6  6  20 20 18 1  19

There are 1 solution(s) for grid #1.
```

```
Grid #2:

5  4  3  6  5  3  4  6
0  6  0  1  2  3  1  1
3  2  6  5  0  4  2  0
5  3  6  2  3  2  0  6
4  0  4  1  0  0  4  1
5  2  2  4  4  1  6  5
5  5  3  6  1  2  3  1


Maps resulting from grid #2 are:

6  20 20 27 27 19 25 25
6  18 2  2  3  19 8  8
21 18 28 17 3  16 16 7
21 4  28 17 15 15 5  7
24 4  11 11 1  1  5  12
24 14 14 23 23 13 13 12
26 26 22 22 9  9  10 10

There are 1 solution(s) for grid #2.
```