

**The Northeast Regional Competition
of the
2002-2003 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 19, 2002**

Problem 1: Golfing USA

Taking the lead from the Ryder Cup, the Eastern Shore Golf Association held its annual North South showdown. Each team had 5 players with the North Team represented by Player1, Player3, Player5, Player7, and Player9. Player2, Player4, Player6, Player8, and Player10 represented the South team.

The format had *Match* competition consisting of 2 players per team and *Individual* competition, which matched each player from each team on a one-to-one basis. Players did not necessarily compete in all of the match play contests. Team scoring is 1 point for a win and 1/2 point for a draw, whether the competition was for Match or Individual play. Individual statistics were also kept [wins-losses-ties-totalPoints] using the same criteria, 1 point for a win and 1/2 point for a draw.

Each line of input for your program contains the following pieces of information: the type of competition indicated by the character M for *Match* or the character I for *Individual*, whether the match was a win or a draw indicated by the character W for a win and the character D for a draw, and numbers representing the players competing. In the case of a win, the winning players are listed before the losing players. There will be a single space between the type of competition, the win/draw indicator and each of the numbers. An input line containing the character X indicates the end of input.

In Team match play, both players on the winning team get credit for a win and are awarded 1 point each in individual statistics. In addition, the winning TEAM is awarded 1 point in team score. Both members of the losing team get credit for a loss in individual statistics. If the match is a draw then all four players get credit for a tie and are awarded 1/2 point each in individual statistics; both TEAMS are awarded 1/2 point in team score.

Output for the program indicates which team won (North or South) or indicates that the match was a draw. In addition, the final team scores should be indicated. The statement should be formatted as one of the following:

"TeamName wins X to Y" or "Match Is A Draw"
The output must also list the players from each team (by most points attained) and their record [wins-losses-ties-totalPoints], as well as indicate which player(s) had the most points in the tournament. (See sample output for format.)

Sample Input:

```
M W 1 3 4 6
M D 2 8 5 9
M D 5 7 2 10
M W 2 4 1 5
M W 9 5 8 4
M W 6 2 5 3
I W 4 7
I W 3 8
I D 1 6
I W 2 5
I W 10 9
X
```

Sample Output:

South wins 6.5 to 4.5

North

Player3	2-1-0-2.0
Player5	1-3-2-2.0
Player9	1-1-1-1.5
Player1	1-1-1-1.5
Player7	0-1-1-0.5

South

Player2	3-0-2-4.0
Player4	2-2-0-2.0
Player6	1-1-1-1.5
Player10	1-0-1-1.5
Player8	0-2-1-0.5

Player2 had the most points in the tournament

**The Northeast Regional Competition
of the
2002-2003 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 19, 2002**

Problem 2: Deciphering bar codes

The United States Postal Service uses an encoding scheme to translate zip codes into bar codes for automatic processing. The bar codes consist of short and long vertical lines, which we will represent by the colon and vertical line characters as shown below:

: for a short vertical line
| for a long vertical line

A hash function is used to assign a unique number to each digit 0 - 9. The hash function is based on the multiplication method and operates as follows: take the digit and add 1, multiply by a given constant A ($0 < A < 1$), extract the fractional part, multiply by a given constant m ($0 < m < 32$) and take the floor. (Note: The constants A and m are chosen so that each of the digits 0 - 9 is assigned a unique number between 0 and 31.) In short, the hash function is

$$h(n) = \lfloor m * ((n+1)*A \bmod 1) \rfloor$$

where "mod 1" represents extracting the fractional part. The resulting hash value is encoded as a binary sequence of length 5, using the character : for 0 and the character | for 1.

The bar code consists of the sequence of encoded hash values for the zip code digits, with a single space between the encoded digits. Following the encoding of the fifth digit is a space and the encoding for a correction digit. The correction digit is computed as follows: add up the digits of the zip code, and choose the correction digit to make the sum a multiple of 10.

For example, the zip code 01129 has correction digit 7 since $0+1+1+2+9 = 13$ and $13 + 7 = 20$. Given $A = .618$ and $m = 31$, the encoding for the zip code 01129 is shown below:

n	(n+1)*A	mod 1	mult by m	floor	encoding
0	0.618	0.618	19.158	19	::
1	1.236	0.236	7.316	7	::
2	1.854	0.854	26.474	26	::
7	4.944	0.944	29.264	29	:
9	6.18	0.180	5.58	5	:: :

$\underbrace{|::||}_{0} \quad \underbrace{::|||}_{1} \quad \underbrace{||::|}_{1} \quad \underbrace{|||:|}_{2} \quad \underbrace{::|:|}_{9} \quad \underbrace{|||:|}_{7}$

Your problem is to write a program that will decode bar codes to produce the equivalent 5-digit zip codes and check the correction digits. The first input line contains the value A for the hash function used in the encoding. The second input line contains the value m for the hash function used in the encoding. The third input line contains the number of zip codes to decode. The remaining lines contain one bar code per line encoded from the hash function with the given input values for A and m. You may assume that each bar code represents an encoded zip code. That is, each binary encoding can only represent one of the ten possible hash values.

Your output should consist of each decoded 5-digit zip code, one per line. If the correction digit for a zip code is incorrect, indicate this fact by printing an asterisk (*) immediately following the zip code. Also print the decoded correction digit for each zip code in column 10 of the output.

Sample Input:

```
.743
29
5
|:|:| :|||: :|||: ::||: :|::: |::||
|:|:: |::|| :|::: ::||: ||::: |:|::
|||:: :|||: ::|:| :|:| |:|:| :|:|
:|::: |::|| |:|:: :|:| :|:| |:|:|
::||: :|||: |||:: ::||: |:|:| ::|:
```

Sample Output:

```
01129* 8
48923 4
31650 5
98455* 0
21320 2
```

**The Northeast Regional Competition
of the
2002-2003 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 19, 2002**

Problem 3: A file packaging problem

Suppose you have a part-time job packaging computer files onto CDs for distribution. You are given bundles of files of various sizes and a supply of CDs with a fixed capacity. Your boss was hoping that you would be able to efficiently package the bundles of files using the least possible (or optimal) number of CDs. But with your great knowledge in computer science, you have informed your boss that, in general, you can only produce an approximate solution. You have, however, assured him that your solution using a "decreasing fit first" approach will guarantee that the number of CDs used will be at most $11/9 * (OPT) + 4$, where OPT is the optimal number. Your "decreasing fit first" approach involves starting a new bundle by placing the file of largest size on a blank CD. Then, iteratively, the next largest file is placed on the CD with the greatest remaining capacity, adding a new CD only in the case that the next largest file will not fit on any of the existing CDs in the bundle.

The first line of the input contains the fixed capacity of a CD - this fixed capacity will be no more than 73,400,320 bytes. (Note: All necessary computations for the program will be no more than 2,147,483,647. In C++, use the long numeric data type; in Java, use the integer numeric data type.) The second line of the input is the number of bundles to be packaged. The remaining lines contain the data for each bundle. The data for a single bundle is formatted as follows: the first line is the number of files in the bundle; each of the remaining lines contains the size of the file, followed by the name of the file beginning at column 10. The maximum length of a filename - including file extension - is 10 characters and the size of any one file does not exceed the fixed capacity. You may also assume that no more than 20 CDs will be necessary for any given bundle.

The output should contain a label for the beginning and end of each bundle and a blank line separates the output for bundles. Output for a bundle contains the number of disks in the bundle and a summary for each disk. Each disk summary contains the number of files contained in the disk, bytes used, bytes remaining, and a directory of the files. The files are listed one per line in order of decreasing size (the order they fit on the disks). Each line begins with a blank space, followed by the name of the file beginning at column 2 and its capacity starting at column 13. The format for the output of a bundle is shown at the top of the next page. (The character # is used to indicate a numerical value and the characters \$\$\$\$ are used to represent filenames.)

```
* * * Bundle 1 * * *
Number of disks: #
Disk # contains # files
Bytes used: #
Bytes remaining: #
Directory:
$$$$$$$$$ #
$$$$$$$$$ #
* * * End of Bundle 1 * * *
```

Sample Input:

```
655360
2
7
5488      hsbr.exe
246581    pict.dat
360448    bigapp.exe
102400    ftprcl.dat
368640    monst.dat
222331    rgpx.dat
105472    angsr.dat
5
163840    mdmap.dat
409600    cplmeg.dat
81920     teth.bmp
65536     wdstc.bmp
174080    bgrap.dat
```

Sample Output:

```
* * * Bundle 1 * * *
Number of disks: 3
Disk 1 contains 2 files
Bytes used: 590971
Bytes remaining: 64389
Directory:
  monst.dat  368640
  rgpx.dat   222331
Disk 2 contains 2 files
Bytes used: 607029
Bytes remaining: 48331
Directory:
  bigapp.exe 360448
  pict.dat   246581
Disk 3 contains 3 files
Bytes used: 213360
Bytes remaining: 442000
Directory:
  angsr.dat  105472
  ftprcl.dat 102400
  hsbr.exe   5488
* * * End of Bundle 1 * * *
```

```
* * * Bundle 2 * * *
Number of disks: 2
Disk 1 contains 2 files
Bytes used: 583680
Bytes remaining: 71680
Directory:
  cplmeg.dat 409600
  bgrap.dat  174080
Disk 2 contains 3 files
Bytes used: 311296
Bytes remaining: 344064
Directory:
  mdmap.dat  163840
  teth.bmp   81920
  wdstc.bmp  65536
* * * End of Bundle 2 * * *
```

**The Northeast Regional Competition
of the
2002-2003 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 19, 2002**

Problem 4: How much can you do in a year?

Given a four-digit year number, find all the different umbers that can be obtained from the four digits by doing addition, subtraction, multiplication, and integer division. The following rules apply:

- 1) The order of the digits cannot be changed.
- 2) All operations have EQUAL precedence and are performed left to right.
- 3) Integer division is truncated towards zero. $9/5$ is 1 and $-9/5$ is -1.
- 4) No parentheses may be used.
- 5) Division by zero is not allowed.

For example, from the year 2002 the following numbers may be obtained:

```
-2    as  2 + 0 * 0 - 2 = 2 * 0 - 2 = 0 - 2 = -2
0     as  2 + 0 + 0 - 2
1     as  2 + 0 + 0 / 2
2     as  2 + 0 * 0 + 2
4     as  2 + 0 + 0 + 2
```

(Notice that 0 is also obtainable as $2 - 0 - 0 - 2$, but it is only listed in the output once.)

The total number of distinct obtainable numbers must be given. Any number of years may be given in the input, one per line. A year number of 0 terminates input and should not be processed. In the output, the individual numbers may be listed in any order, one per line.

Sample input

```
2000
0
```

Sample output

```
For the year 2002 there are 5 values.
-2
0
1
2
4
```

**The Northeast Regional Competition
of the
2002-2003 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 19, 2002**

Problem 5: Undo!

An inversion of an array is a pair of elements that are out of order. For example, if an array is to be sorted to ascending order, then the array

20, 5, 10, 9, 15

has 5 inversions:

20, 5
20, 10
20, 9
20, 15
10, 9

Give a set of inversions, we may also be able to find its original unsorted array. For example, if an array is to be sorted to

5, 9, 10, 15, 20

and it has 3 inversions:

10, 9
20, 9
20, 15

then 10 is on the left of 9, 20 is on the left of 9 and 15, and all other relative orders are not changed. So, the original array must be

5, 10, 20, 9, 15

However, there are sets of inversions for which the corresponding unsorted arrays can not be determined. For example, if an array is to be sorted to

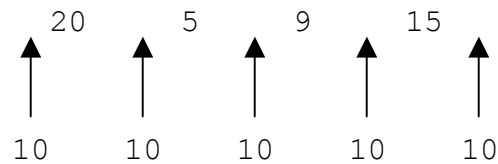
5, 9, 10, 15, 20

then it can not possibly contain only the following 3 inversions:

20, 5
20, 9
20, 15

because it must contain some other inversion(s), no matter where

the number 10 is:



Your program is to determine the corresponding unsorted array from a given set of inversions. You will be processing an unknown number of arrays. However, all arrays are of the same size, and all are to be sorted in ascending order to the same result. You may assume that the maximum size of each array is 20, and all numbers in the array are distinct integers.

The first input line contains a positive integer indicating the size of each array. The second data line contains the sorted array. Starting the third line, there is an unknown number of inversion sets, with one inversion per line. Each set of inversions is terminated by a dummy data line containing two zeros. For each set of inversions, print the corresponding unsorted array, or the message "no matching array" if no such array can be found. Continue to process each set of inversions until a dummy data line containing two -1's. Be sure to print a blank line between two output lines.

Sample Input

```
5
5 9 10 15 20
20 5
20 10
20 9
20 15
10 9
0 0
10 9
0 0
0 0
20 5
20 9
0 0
20 9
10 9
20 15
0 0
-1 -1
```

Sample Output

```
20 5 10 9 15
5 10 9 15 20
5 9 10 15 20
no matching array
5 10 20 9 15
```

Note that only sorted arrays does not any have inversion. In the sample input above, the third set contains no inversion, so the sorted array is printed.

**The Northeast Regional Competition
of the
2002-2003 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 19, 2002**

Problem 6: XXL Precision

In 1994, the Intel Corporation introduced its new Pentium chip. It was soon discovered that the Pentium chip only provided single-precision accuracy on floating-point arithmetic. The news was published on the Internet and the front page of the world's newspaper. The Intel traced the problem to 5 missing entries in a lookup table used to implement the radix-4 SRT algorithm of the Pentium's division instruction. A quick fix of the problem before the new chip can be manufactured is to provide a software solution. Your program is to do that. You are to write a program that can perform integer division with XXL (extra extra long) precision. In fact, it can produce accuracy to any desired decimal place. Each input line to your program will contain 4 positive integers: a , b , m and n , where $m \leq n$. Your program should divide a by b , and print the m th through n th positions of the result. Here are some examples:

- (1) If $a = 7$, $b = 5$, $m = 1$ and $n = 5$, then the output should be

position 1 to position 5 of $7/5 = 1.400$

because $7/5 = 1.40000000\dots$. Note that each digit and the decimal point is considered as one position. So the second position of $7/5$ is the decimal point, and its third position is 4.

- (2) If $a = 500$, $b = 7$, $m = 10$ and $n = 20$, then the output should be

position 10 to position 20 of $500/7 = 42857142857$

because $500/7 = 71.42857142857142857142\dots$

- (3) If $a = 50000$, $b = 7$, $m = 5$ and $n = 8$, then the output should be

position 5 to position 8 of $50000/7 = .857$

because $50000/7 = 7142.857142857142857142\dots$

- (4) If $a = 2$, $b = 35$, $m = 5$ and $n = 5$, then the output should be

position 5 to position 5 of 2/35 = 7

because $2/35 = 0.057142\dots$

Your program should continue to read and process until a line containing a single zero is reached. Note that this zero is a sentinel, and it should not be processed.

Note that each output line must be in the format:

position ### to position ### of ###/### = ###

where ### is a number. You should also print a blank line between two output lines.

Sample Input:

```
7 5 1 5
500 7 10 20
50000 7 5 8
2 35 5 5
0
```

Sample Output:

```
position 1 to position 5 of 7/5 = 1.400
position 10 to position 20 of 500/7 = 42857142857
position 5 to position 8 of 50000/7 = .857
position 5 to position 5 of 2/35 = 7
```

**The Northeast Regional Competition
of the
2002-2003 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 19, 2002**

Problem 7: A Very Dicey Situation

Ada Lovelace's kid sister has been stacking standard dice. She puts the dice in four equally high stacks and arranges them in a 2x2 square. The dice are oriented based on input data from Ada's latest program.

Ada decides to play a game. She removes dice from two adjacent stacks and rotates the dice on the top of the other two stacks 90 degrees away from the stacks where the dice were removed (so that the former top face is now facing away from the adjacent stack that was removed). She wants to know if it is possible to remove dice in such a way that all four stacks have ice with the same number (e.g. 1, 2, 3, 4, 5, or 6) showing on top (not necessarily at the same height), given the initial setup. The dice removed at each stage are not necessarily at the same height, but merely in adjacent stacks.

The first line of input is the common height of the four stacks. It will be a one-digit number followed immediately by an end-of-line. If this number is n , then n lines of input follow that list the dice at the same height (beginning at the bottom). Each die is represented by two numbers between 1 and 6 inclusive. The first is the number on top and the second is the number on the right (or "east") side. All dice in the judges' data are standard dice, so these two numbers never add up to 7. Four dice will appear on a single line of input (e.g. eight numbers), each number being one digit. Each number except the last is followed by exactly one blank space and the last number is followed immediately by end-of-line. The dice are in row-major order, so the first two dice (four numbers) represent the top row of dice and the last two dice (four numbers) represent the bottom row of dice on that level.

This pattern of input is repeated until the height is 0. The 0 marks the end of the input and should not otherwise be processed.

The output should indicate the common number on top of the stacks, if one can be found, and a statement that you have won the game. If more than one common number is possible, all should be listed. If no common number can be found, the output should simply say that you have lost the game. This output should be repeated for each set of input data with positive height.

Sample input:

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

Sample output:

```
The winning number is 1
You won the game!
```

```
You lost the game!
```



Note that a common number of 1 in the first set of data is achieved by removing the top row of dice, exposing two 1's on the bottom level in the top row and rotating the two dice on top of the bottom row at the 2nd level so that 1's show on top.