1998 APICS Programming Contest Problems

The 1998 Atlantic Canada programming Contest was held at St Marys University in Halifax Nova Scotia Friday Oct 16 1998. There were 18 teams of 3 students from ten universities in the four Atlantic provinces.

Each team of 3 students was given 6 problems to try to solve in 5 hours using one computer following the rules for the ACM programming contests.

Each problem has a set of associated data files which are used one at a time by both the programmers and the judges to test the program.

The participating teams came from the folowing universities:

```
University of New Brunswick, Fredericton Campus - 3 teams
University of New Brunswick, St John     Campus - 1 team
University de Moncton - 2 teams
Mount Allison Univ., Sackville N.B. - 2 teams

Acadia Univ., Wolfville N.S.  - 1 team
Dalhousie Univ. Halifax N.S.  - 3 teams
St Marys Univ. Halifax  N.S.  - 1 team
St Francis Xavier Univ.,N.S.  - 2 teams

Univ. of Prince Edward Island - 1 team

Memorial Univ. Newfoundland   - 1 team
```

The top three teams from this competition went on to the next level of competition, Oct. 24 in Westfield Massachusetts.

The decoding problem (Porter Scobey)

The program you are to write for this problem must decode the message in an encoded input textfile infile.dat containing any number of lines of exactly 60 numerical digits each, and print the plain text message content of the message to standard output. There are two sample textfiles containing input data called decode1.dat and decode2.dat both of which contain an encrypted message.

There is no size limit on a message, so your program must be able to decode an encoded textfile of any size.

We now describe the encoding scheme used.

Each character in the plain-text message is examined and its ASCII code is determined. A value of 100 is added to the ASCII code and the result is then multiplied by 3. The resulting (three-digit) integer is then written out to the output file as the encoded form of that character. The end-of-line character is treated as having ASCII code 13 and is otherwise encoded like any other character.

One other thing. Note that every line in an encoded file contains exactly 60 digits. Clearly this means the last line as well, and just as clearly, then, there may be some "garbage" digits filling out the last line just to make up the full 60 characters. If so, the start of the garbage is indicated by the triplet 975 (which would otherwise be invalid), and both it and all of the remaining characters on the last line should be ignored in the output.

Sample input file (decode1.dat) with 7 lines

```
549633627603594633600663396636624603591645603396627591621603
396645651642603396648612591648339648612615645396615645396627
663396603636615648591636612474339396396396396519606396519396
621630603657396648612603630396657612591648396519396621630633
657396630633657432339396396396396552612591648396657633651624
600396612591654603396627603645645603600396615648396651636396
645633627603612633657399339975626040634742010082632270616415
```

Decrypted output for the above input file(4 lines)

```
Somebody please make sure that
this is my epitaph:
    If I knew then what I know now,
    That would have messed it up somehow!
```

Write your program to read input from file infile.dat and copy either decode1.dat or decode2.dat to infile.dat before running your program. The judge may use another data file.

The word puzzle problem (Ke Qiu)

In this problem you are given an n by n array of letters, and a list of words. Note that a word is not necessarily a proper word that you can find in a dictionary. The program is to determine if any word from the list can be found in the array, and if so, find positions of its beginning and ending letters. A word can appear in the array in three ways: horizontally, vertically, and diagonally, each in both directions (there is no wrap-around). A word can also appear in the array more than once.

Here is a sample input file (puzzle1.dat):

```
5abcedgratoitboyeargotrace
3bra3cab3eat3ate3boy3rae5trace4race3rat6simple5great2go0
```

Note that the first line has a number n followed by n times n letters in this case representing the array

```
a b c d e
g r a t o
i t b o y
e a r g o
t r a c e
```

Note that the second line in the input file contains a number of words each preceded by the number of characters in the word. The last word is immediately followed by a zero. The words in the example above are bra, cab, ..., go.

The output of your program for the above input file should look like the following although the spacing does not have to be exactly like this.

```
Word Beginning Ending
bra  (3, 3)   (1, 1)
cab  (1, 3)   (3, 3)
...
rat  (5, 2)   (3, 2)
go   (4, 4)   (4, 5)
```

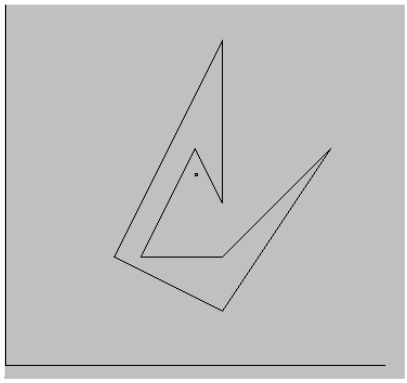
Write your program to read from file infile.dat. Copy one of the given data files puzzle1.dat or puzzle2.dat to infile.dat before running your program. The judge may run your program on other data files.

The point in polygon problem (Ke Qiu)

In a Graphic User Interface (GUI), a program is depicted by an icon, that is usually a simple polygon (not necessarily convex), and the program is activated by clicking the mouse when the mouse pointer is inside the icon. Therefore, to decide whether to activate a particular program when the mouse is clicked, we first check whether the mouse pointer is inside the polygon.

Write a program such that when given a simple polygon P and a series of points z=(x, y), it can determine whether each point z is inside P. Thus, your program should answer YES if z is inside P and NO otherwise. A sample input file (polygon1.dat) is shown below:

```
8
100 100
200 300
200 150
175 200
125 100
200 100
300 200
200  50
175 175
125 100
```



In general, the input file has a number n on the first line followed by n lines each containing an (x,y) vertex of the polygon and the remaining lines each contains a point z = (x, y) which you are to check to see if it falls strictly inside the polygon. The polygon is a closed region formed by drawing a line from each point to the next one and closed with an edge connecting the last (n'th) point back to the first one.

You may assume that the polygon is simple - no edge crosses another. However it need not be convex. See the diagram of the 8 points above where the origin is at the bottom left. The polygon may not even have any interior points!

Your program should output each of the points followed by the word YES or NO as appropriate. For the sample input file above the output would be

```
175 175 NO
125 100 NO
```

since the first point is not inside and the second is on the border.

It may help to recall that a point (x,y) lies on a line joining (a,b) and (c,d) if (y-b)/(x-a) = (d-b)/(c-a). Also recall that if you have two equations ax + by = c, dx + ey = f the solution is x = (bf-ce)/(bd-ae), y = (cd-af)/(bd-ae).

Write your program to read from file "infile.dat". Copy polygon1.dat or polygon2.dat to file infile.dat before running your program. The judge may use another data file.

The Rectangle problem (Andrew Rau-Chaplin, Arthur Sedgwick)

In this problem you are given a set of rectangles and you are to calculate the area of the union of the rectangles. Note that where rectangles overlap the area of the overlapping region is only to be counted once.

A sample data file (rect1.dat) is

```
50 50    100 100
125 125   75 75
60 90     90 60
```

The output for this data file should be

```
The area is 4375
```

In general the data file contains any number of lines each containing two pairs of integer values. These pairs represent two diagonally opposite points defining a rectangle with sides that are all horizontal or vertical. Write your program to read input from file infile.dat and copy either rect1.dat or rect2.dat to infile.dat before running your program. The judge may use another data file.

Most solutions to this problems did not work with all data sets. One approach that works is as follows. At all times we have a list of disjoint rectangles. When we read in a rectangle B we go through the list of previous disjoint rectangles and subtract B from each one that it overlaps. When one rectangle B overlaps another A you subtract B from A leaving up to 4 smaller rectangles from A which are disjoint from B.

The KWIC index problem (Arthur Sedgwick)

A Keyword-In-Context or KWIC index is an alphabetical listing of all the words in a group of sentences with each word printed on a separate line with the part of the sentence in front of the word and the part after also printed to give the "context". It is often used in libraries where it allows you to quickly find all titles containing a certain word.

The following sample input file (kwic1.dat) has three sentences and 14 "words" (sequences of non-blank characters).

```
All's well that ends well.
Nature abhors a vacuum.
Every man has a price.
```

Here is the output (KWIC index) for this input:

```
  Every man has a price.
  Nature abhors a vacuum.
         Nature abhors a vacuum.
                All's well that ends well.
All's well that ends well.
                Every man has a price.
      Every man has a price.
          Every man has a price.
                Nature abhors a vacuum.
Every man has a price.
     All's well that ends well.
Nature abhors a vacuum.
          All's well that ends well.
All's well that ends well.
```

Note that the number of lines in the output should be the same as the number of "words" in the input. Note that there is a column (in this example under the fourth word in the first output line) which contains all the words in alphabetical (lexicographic) order disregarding whether the letters are upper- or lower-case.

Note that blanks are printed at the beginning of most lines so that the "words" line up this way. However, your program should not print any unnecessary blanks. At least one output line should have no leading blanks (This happens to be the last output line in the example.)

If the input sentences all have just one word then the output should consist of the words in alphabetical order (as above) with no leading blanks on each line.

Note that extra blank characters at the beginning or end of the sentences should be removed.

Write your program to read from file infile.dat. Copy kwic1.dat or kwic2.dat to infile.dat before running your program. The judge may use another data file.

Lost in the Maze (Patricia Evans)

A simple automaton has become lost in a grid maze. It is trying to get to a particular destination square, but needs your help. The automaton only has three possible actions (f,l,r): it can move one square forward (f), it can turn 90 degrees to its left (l), or it can turn 90 degrees to its right (r). Each action takes up one move.

Determine the shortest list of moves that will get the automaton from its current location and orientation to its destination. If there is more than one shortest list any one is acceptable.

The input consists first of one line containing two integers: the number of columns and the number of rows. The rest of the input is the rows of the grid. Each square is represented by one character. An x represents a solid square that cannot be travelled through, while a period (.) represents an open square that can be travelled through. The destination square is represented by an asterisk (*). The position and orientation of the automaton is represented by one of four characters (>,<,v,^), and points in the direction that the automaton is facing. Both the destination square and the square currently occupied are considered open.

Example input (maze1.dat):

```
7 5
x...xx*
..x..x.
.xxx...
..v.xxx
x..xxxx
```

The output consists of the shortest number of moves followed by the list of those moves.

Example output:

```
25 moves:
rffrffrflfrffrflfrflfflff
```

You may assume that there is solution to the problem.

Write your program to read from file infile.dat. Copy maze1.dat or maze2.dat to file infile.dat before running your program. The judge may use another input file.