NMU PROGRAMMING CONTEST #1
Saturday 1 April 2000

1. A team consists of at most three members. Each team will have a single computer at its disposal to develop solutions to the assigned problems. The team may share use of the computer in any way it sees fit to do so. The team may solve the problems individually or collectively at its discretion. However, a team member may not communicate--personally, electronically, or otherwise--with anyone other than a judge or another member of the same team during the contest.

2. The judges will be happy to answer questions of clarification during the contest and are willing to assist with technical difficulties, but they cannot provide any hints or information relating to how the problems are to be solved.

3. There are 6 contest questions and 5 hours in which to solve them. The questions need not be of equivalent difficulty with each other and are given in no particular order. They may be solved in whatever order the team sees fit. For each problem, the team is to generate a single file, an executable called **prob*.exe** , where * is the digit between 1 and 6 corresponding to the problem being solved. **prob*.exe** will open an existing file, **prob*.in** , for reading, and will create and open a file, **prob*.out** , for writing. The team may use any language of its choice to develop **prob*.exe** , and is not even constrained to using the same high-level language for each problem. The team may use any printed reference materials during the duration of the contest but is disallowed from searching the Internet for assistance. **prob*.exe** must use the input and output file described above and no other files. The use of other files will cause the program to fail during judicial testing. Any output sent to the screen will be ignored by the judges, only the output sent to **prob*.out** , will be evaluated. The judge will not enter information to the program via the keyboard, so any attempt to read from the keyboard will cause the program to hang during judicial testing, thus disqualifying it, as described below.

4. **prob*.in** and **prob*.out** are text files. They will be constrained to containing only the 94 visible ASCII characters, the space character, and the end-of-line character, which varies between systems and will be denoted by **<EOLN>**. In particular, a problem will never ask you to parse or print a **<TAB>** character. The files will be limited to at most 10000 characters. No input file will exceed this length, and no output file exceeding this length will be required. The team may assume that the judicial input file will be formatted correctly--it is not necessary to check for syntax errors--and the output file must be formatted exactly to specification. For each valid input file, there will be one and only one correct output file. Each problem specification will contain a sample input and output file, but this sample need not be the same input file used for judicial testing. For specification purposes, the end-of-file marker will be denoted by **<EOF>**.

5. When a solution has been developed for a particular problem, a member of the team will deliver the appropriate executable on a floppy disk to a judge, who will mark the time of receipt. The member should indicate to the judge which solution is being attempted. The floppy must contain the appropriate executable, but no other file on the floppy will be examined. In particular, it is not necessary to include the source code on the floppy. The judge will execute **prob*.exe** against a preexisting **prob*.in** file. The generated **prob*.out** will be compared against the one and only correct output file. The team will be told whether it has successfully generated the correct output file, but, in the event that the output file is incorrect, no information beyond the fact that it is incorrect will be revealed to the team. Any executable that takes longer than 5 seconds to execute will be considered an incorrect solution, but, again, the only information revealed to the team will be that the solution was incorrect.

6. Any attempt by **prob*.exe** to copy or damage the preexisting input file in any fashion or to demean the integrity of the contest in any way will result in immediate disqualification of the

submitting team.

7. Scoring is determined by a pair of integers as follows: The **count** for each team is the number of problems solved. The **total** for each team is the total number of minutes required to solve each problem starting from the beginning of the contest. A 20 minute penalty is added to the **total** for each unsuccessful attempted solution of a problem, provided that a correct solution of that problem is eventually submitted. More precisely: at the beginning of the contest, **count** and **total** are initialized to zero. Upon the evaluation of a successful executable, **count** is incremented by one, **total** is incremented by the number of minutes between the start of the contest and the submission of the executable, and **total** is further incremented by 20 for each incorrect solution to the problem submitted prior to the correct one.

8. Ranking is determined by each team's **count**, with the highest **count** being ranked first. Ties are broken by **total**, in favor of the team with the lowest **total**. Any subsequent ties remain unbroken. With this in mind, it is to the team's advantage to seek out and complete the easier problems first. This will result in a smaller **total**. However, keep in mind as well that difficulty is a matter of individual perception, and that the problems are given in no particular order.

REMINDERS

1. When a program fails, you will only be told that it failed, you will not be told why. You may have made a simple careless error like those described below.

2. Failure to format the output file EXACTLY as specified will cause the program to fail.

3. Failure to name the executable, input, and output files EXACTLY as specified will cause the program to fail.

4. Reading input from the keyboard will cause the program to fail.

5. Screen output will be absolutely ignored.

6. You will be penalized for programs that fail.

7. <EOLN> and <EOF> in the specifications refer to your computer's end–of–line and end–of–file markers. They do not represent the literal strings "<EOLN>" and "<EOF>".

Problem 1: Mark Terwilliger's Word Search

This problem involves those classic word search puzzles that you did when you were a kid (or maybe still do). The input will consist of several word search puzzles, each one consisting of a square grid of letters and a list of words to find. In this particular problem, your program should search for words up, down, left, and right. If a word is not found, a message should reflect this. You may assume that no word will be found more than once.

INPUT SPECIFICATION:

The input data will contain an arbitrary number of test cases, followed by 0**<EOLN>**. Each test case will have the following format:

A decimal integer N, between 5 and 10 inclusive followed by **<EOLN>**, representing the dimensions of the grid. Following this are N rows of N capital letters. There are no spaces in the row, and each row is terminated by **<EOLN>**.
Following this is a decimal integer M, between 1 and 15 inclusive followed by **<EOLN>**, representing the number of words to search for.
Following this are M words, each word containing between 3 and 8 capital letters inclusive. There are no spaces within the words and each word is terminated by **<EOLN>**.

There may or may not be one or more additional **<EOLN>**'s separating the test cases, or between the final test case and the terminating 0, but no superfluous **<EOLN>** will appear within a test case.

OUTPUT SPECIFICATION:

The test cases should appear in the output in the order in which they appear in the input. The output for each test case will consist of M lines (M defined above). Each line should begin with the word being searched for (and these words should appear in the output in the order in which they appear in the input). If the word is present in the grid, the word should be followed by the column number and the row number of the first letter in the word (the upper left corner is row 1, column 1) and the direction (U, D, L, R) in which the word is spelled out. There should be exactly one space between word and column, column and row, and row and direction, and the line should be terminated by **<EOLN>**. If the word is not present in the grid, the word should be followed by " WAS NOT FOUND**<EOLN>**". Note that there is exactly one space between all words in this line. There should be one extra **<EOLN>** following each test case in the output.

SAMPLE INPUT

```
5<EOLN>
ABCDE<EOLN>
HORSE<EOLN>
XYZAB<EOLN>
WOCQU<EOLN>
FOXES<EOLN>
4<EOLN>
HORSE<EOLN>
COW<EOLN>
DOGS<EOLN>
BEE<EOLN>
<EOLN>
5<EOLN>
ELLPA<EOLN>
APLLA<EOLN>
ALLPE<EOLN>
PALEA<EOLN>
ELPPA<EOLN>
3<EOLN>
APE<EOLN>
PEEL<EOLN>
APPLE<EOLN>
<EOLN>
0<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
HORSE 1 2 R<EOLN>
COW 3 4 L<EOLN>
DOGS WAS NOT FOUND<EOLN>
BEE 5 3 U<EOLN>
<EOLN>
APE 1 3 D<EOLN>
PEEL WAS NOT FOUND<EOLN>
APPLE 5 5 L<EOLN>
<EOLN>
<EOF>
```

Problem 2: Barry Peterson's Sentence Spiral

Although the English language is a left–to–right, top–to–bottom language, this is not the only way to orient text on a page. Hebrew, for example, is right–to–left, top–to–bottom, and traditional Chinese is top–to–bottom, right–to–left. Well, in the mythical country of Yooperland, text is arranged in a counter–clockwise spiral starting from the lower–left corner of a square grid and ending in a central position. For example, the phrase "SECOND WEEK OF DEER CAMP!" can be written in a 5 x 5 grid as follows:

```
O KEE
FMACW
 P!
DEERD
SECON
```

For this problem you will take strings of text written in the standard English left–to–right format and print them out in the square spiral format of Yooperland.

INPUT SPECIFICATION

You will be given a series of test cases, followed by 0**<EOLN>**. Each test case will begin with a decimal integer N between 4 and 10 inclusive followed by **<EOLN>**. The next line will contain a string containing no more than N^2 characters and terminated by **<EOLN>**. There may or may not be one or more additional **<EOLN>** characters between test cases or between a test case and the terminating 0.

OUTPUT SPECIFICATION

The test cases should appear in the output in the order in which they appeared in the input. For each test case, you should print an N x N grid of letters, each line terminated by **<EOLN>**. If the input string is smaller than N^2 characters in length, the string should be padded on the right with spaces to N^2 characters before being placed in the grid. Aside from the trailing spaces and the spaces which were in the input string originally, no extra spaces should appear in the output. Each test case should be followed by an extra **<EOLN>** character.

SAMPLE INPUT

```
5<EOLN>
SECOND WEEK OF DEER CAMP!<EOLN>
<EOLN>
6<EOLN>
Hockey determines a school's merit<EOLN>
<EOLN>
0<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
O KEE<EOLN>
FMACW<EOLN>
 P!  <EOLN>
DEERD<EOLN>
SECON<EOLN>
<EOLN>
enimre<EOLN>
sm s't<EOLN>
 e  le<EOLN>
aritod<EOLN>
 scho <EOLN>
Hockey<EOLN>
<EOLN>
<EOF>
```

Problem 3: Phil Sweany's Consecutive Sum

A consecutive sum is a sum of at least two consecutive positive integers. 4+5+6 is a consecutive sum leading to 15, for example. For any given positive integer, you are to determine EVERY consecutive sum leading to that integer. For example, 15 can be attained by 1+2+3+4+5, by 4+5+6, and by 7+8.

INPUT SPECIFICATION

The input file will contain a series of test cases, followed by 0**<EOLN>**. Each test case is a decimal integer between 1 and 1000000 inclusive, followed by **<EOLN>**.

OUTPUT SPECIFICATION

The test cases should appear in the output in the order in which they appear in the input. For each input N, every consecutive sum leading to N must be printed. DO NOT print out every number in the sum. Each sum should be printed out in the following manner: the lowest number in the sum, one space, the highest number in the sum, **<EOLN>**. If a test case generates more than one consecutive sum, they must be printed in order of length, starting with the longest sum and ending with the shortest sum. After all consecutive sums for a test case has been printed, an extra **<EOLN>** should be printed following them. If a number has no consecutive sum, then no consecutive sum can be printed, but the extra **<EOLN>** described in the previous sentence should still be present.

SAMPLE INPUT

```
15<EOLN>
10000<EOLN>
0<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
1 5<EOLN>
4 6<EOLN>
7 8<EOLN>
<EOLN>
18 142<EOLN>
297 328<EOLN>
388 412<EOLN>
1998 2002<EOLN>
<EOLN>
<EOF>
```

Problem 4: Jeff Horn's Robots

A robot needs to travel from the NW corner of a room to the SE corner. It can't walk there directly, though, because of obstacles in its path. You are to write a program to find the SHORTEST path the robot can take from corner to corner while avoiding all obstacles in its path.

The robot can can move N, S, E, or W, but not diagonally, and must remain at all times within the boundary of the room. You are to print out the shortest path that the robot can take to get from corner to corner.

INPUT SPECIFICATION

There will be a series of test cases, followed by 0 0**<EOLN>**. Each test case will begin with two decimal integers between 2 and 10 inclusive, separated by a space and followed by **<EOLN>**. These numbers, M and N respectively, define the size of the room: M rows and N columns. What will then follow are M lines of N characters each. Each line contains only the characters 0 and 1, and each line is terminated by **<EOLN>**. 0 corresponds to an open space, and 1 corresponds to an obstacle. The first character in the first line (the NW corner) and the last character in the last line (the SE corner) will always both be 0. There will always be at least one path from the NW corner to the SE corner, and while they may be many paths between them, the shortest path will always be unique. There may or may not be one or more additional **<EOLN>** characters separating the test cases and separating the final test case from the terminating 0 0, but there will be no extra spaces or **<EOLN>** characters appearing within a test case beyond that described above.

OUTPUT SPECIFICATION

The test cases in the output file should appear in the order in which they appear in the input file. Each test case in the output file will be a string of characters from among {N,S,E,W} corresponding to the path the robot has to take to get from the NW corner to the SE corner. The first letter in the string corresponds to the direction in which the robot takes its first step, and so on. This string should contain no spaces and should be terminated by **<EOLN>**. No extra **<EOLN>** characters should appear in the output file.

SAMPLE INPUT

```
4 4<EOLN>
0111<EOLN>
0011<EOLN>
1011<EOLN>
0000<EOLN>
<EOLN>
7 6<EOLN>
000000<EOLN>
110001<EOLN>
100011<EOLN>
101011<EOLN>
101011<EOLN>
101110<EOLN>
100000<EOLN>
<EOLN>
0 0<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
SESSEE<EOLN>
EESSWSSSSEEEE<EOLN>
<EOF>
```

Problem 5: Andy Poe's Tic–Tac–Toe

Everyone is familiar with the children's game of Tic–Tac–Toe. One player is X, the other player is O. The players in turn (X goes first) place their mark in a previously unoccupied square of a 3 x 3 grid. A player who gets three of his marks in a row, across, down, or diagonally, wins the game. There are 8 ways to win. If the board is filled without either player getting 3 in a row, the game "goes to the cat" and there is no winner.

Given a valid configuration of the board at some point during a game, you are to determine, assuming both players make their best possible moves for the remainder of the game, which player will win or whether the game will go to the cat.

INPUT SPECIFICATION

The input file will begin with a decimal integer N between 1 and 100 inclusive, followed by **<EOLN>**, representing the number of test cases to follow. What will then follow are N lines, each 9 characters long, and each terminated by **<EOLN>**. These 9 characters will be from {X,O,B}, representing whether an X or O occupies the square or whether the square is blank. The squares are listed in this order: the top row (left to right) followed by the middle row (left to right) followed by the bottom row (left to right). You may assume the board is a valid configuration of an incomplete game of Tic–Tac–Toe. There will be no extra spaces or **<EOLN>** characters in the file beyond those described above.

OUTPUT SPECIFICATION

The test cases in the output should appear in the order in which they appear in the input. For each test case you are to print one of the following strings: "X wins the game.", "O wins the game.", or "The game goes to the cat." depending on the output of the game. Each line should appear EXACTLY as specified and should be terminated by **<EOLN>**. No extra **<EOLN>** characters should appear in the output file.

SAMPLE INPUT

```
2<EOLN>
XOBBBBBBB<EOLN>
BBBBBBBBB<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
X wins the game.<EOLN>
The game goes to the cat.<EOLN>
<EOF>
```

Problem 6: Randy Appleton's Plane

A woman is walking on an infinite plane (a mathematical plane, not an airplane) divided into squares. She can move one square to the north, south, east, or west with each step. You will be given the walk this woman takes on this plane. You are to print out the number of distinct squares (including her starting square) that she visited on the walk. A square only counts once in the count, no matter how many times it has been visited (provided that it HAS been visited).

INPUT SPECIFICATION

The input file will begin with a decimal integer N between 1 and 100 inclusive, followed by **<EOLN>**, representing the number of test cases that will follow. What will then follow are N lines, corresponding to the test cases. Each line will contain a string of characters from {N,S,E,W} of length between 0 and 80 inclusive, followed by **<EOLN>**. This string corresponds to the steps the woman takes as she walks along the plane. There will be no spaces anywhere in the file and no extra **<EOLN>** characters beyond those described above.

OUTPUT SPECIFICATION

The test cases in the output file should appear in the order in which they appear in the input file. Each test case in the output file will be a positive decimal integer, followed by **<EOLN>,** equal to the number of squares the woman has visited. No extra **<EOLN>** characters should appear in the output.

SAMPLE INPUT

```
2<EOLN>
SSSSSSSSSS<EOLN>
NSEWNSEWNSEWNSEWNSEWNSEW<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
11<EOLN>
3<EOLN>
<EOF>
```