Notre Dame ACM Programming Contest

Saturday April 8, 1995

Problem 1
"Pascal's Triangle of Death"

Description:

In this problem, you are asked to generate Pascal's Triangle. Pascal's Triangle is useful in many areas from probability to polynomials to programming contests. It is a triangle of integers with "1" on top and down the sides. Any number in the interior equals the sum of the two numbers above it. For example, here are the first 5 rows of the triangle.

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

In "Pascal's Triangle of Death," you are to generate a **left justified** Pascal's Triangle. When any number in the triangle is *exceeds* or *equals* 1,000,000, your program should finish printing the current row and exit. The output should have each row of the triangle on a separate line with one space between each element. The final element of each line should be directly followed by a newline. There is no space after the last number on each line.

Sample Input:

There is no input for this problem.

Sample Output:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
.
.
.
```
etc.

Problem 2
“English-Number Translator”

Description:

In this problem, you will be given one or more integers in English. Your task is to translate these numbers into their integer representation. The numbers can range from negative 999,999,999 to positive 999,999,999. The following is an exhaustive list of English words that your program must account for:

```
negative, zero, one, two, three, four, five, six, seven, eight,
nine, ten, eleven, twelve, thirteen, fourteen, fifteen, sixteen,
seventeen, eighteen, nineteen, twenty, thirty, forty, fifty,
sixty, seventy, eighty, ninety, hundred, thousand, million
```

Notes on input:
1. Negative numbers will be preceded by the word negative.
2. The word “hundred” is not used when “thousand” could be. For example, 1500 is written “one thousand five hundred”, **not** “fifteen hundred”.
3 The words “hundred,” “thousand,” and “million” are always immediately preceded by a word indicating the numbers one through nine, inclusive.

The answers are expected to be on separate lines with a newline after each.

Sample Input:

```
six
negative seven hundred twenty nine
one million one hundred one
```

Sample Output:

```
6
-729
1000101
```

Problem 3
"Boggle Blitz"

Description:

In the game of Boggle, you are presented with an *NxN* table of letters. The object is to find words in the mix of letters. A word may be started anywhere in the table and is constructed by forming a chain of adjacent letters. Adjacent means diagonal, vertical or horizontal. A word cannot use any character from the table more than once.

Here is an example of a 4*x*4 table:

```
bile
tglp
aest
here
```

The following is a partial list of legal words that can be found using the above rules:

```
bill gates slept here
```

In "Boggle Blitz" you will be given an integer number, *N*, on a single line. *N* is the number of characters on each side of the table, giving a total of *N***N* characters. *N* can range from 1 to 20. *N* lines of *N* characters each will follow giving you the arrangement of the table. Your task is to find all the legal words in the table. **Be sure that your program is efficient!**

In case you forgot to bring a dictionary, you are in luck because we are not using English words. Instead, we have redefined "word" to mean an increasing (by ASCII value) chain of characters from length 3 to length *N***N*. "ABCDE" is a legal five letter word. "MICROSOFT" is **not legal** because the sequence is not increasing. "BILL" is also illegal because L is not greater than L. "BIL," however, is legal.

Your program should output the list of unique words sorted according to the following criterea:

1) Shorter words are before longer words
2) Words are sorted lexigraphically by ASCII value

Add no blank lines or spaces. If no legal words can be found, print nothing.

Sample Input 1:

```
3
one
top
dog
```

Sample Output 1:

```
dop
dot
eno
enp
ent
eop
eot
gop
got
nop
not
enop
enot
```

Sample Input 2:

```
4
abcd
bcda
cdab
dabc
```

Sample Output 2:

```
abc
abd
acd
bcd
abcd
```

Problem 4
"Triangle Wave"

Description:

In this problem you are to generate a triangular wave form according to a specified pair of Amplitude and Frequency.

The input will contain two integers, each on a separate line. The first integer is the Amplitude; the second integer is the Frequency.

For the output of your program, you will be printing wave forms each separated by a blank line. The total number of wave forms equals the Frequency, and the horizontal "height" of each wave equals the Amplitude. The Amplitude will never be greater than nine.

The waveform itself should be filled with integers on each line which indicate the "height" of that line.

NOTE: There is a blank line after each separate waveform, **excluding** the last one.

Sample Input:

```
3
2
```

Sample Output:

```
1
22
333
22
1

1
22
333
22
1
```

Problem 5
"Hangman Judge"

In "Hangman Judge," you are to write a program that judges a series of Hangman games. For each game, the answer to the puzzle is given as well as the guesses. Rules are the same as the classic game of hangman, and are given as follows:

1. The contestant tries to solve to puzzle by guessing one letter at a time.

2. Every time a guess is correct, all the characters in the word that match the guess will be "turned over." For example, if your guess is "o" and the word is "book", then both "o"s in the solution will be counted as "solved."

3. Every time a wrong guess is made, a stroke will be added to the drawing of a hangman, which needs 7 strokes to complete. Each unique wrong guess only counts against the contestant once.

```
    ______
   |   |
   |   O
   |  /|\
   |   |
   |  / \
 __|_
|   |______
|__________|
```

4. If the drawing of the hangman is completed before the contestant has successfully guessed all the characters of the word, the contestant loses.
5. If the contestant has guessed all the characters of the word before the drawing is complete, the contestant wins the game.
6. If the contestant does not guess enough letters to either win or lose, the contestant chickens out.

Your task as the "Hangman Judge" is to determine, for each game, whether the contestant wins, loses, or fails to finish a game. Your program will be given a series of inputs regarding the status of a game. All input will be in lower case. The first line of each section will contain a number to indicate which round of the game is being played; the next line will be the solution to the puzzle; the last line is a sequence of the guesses made by the contestant. A round number of -1 would indicate the end of all games (and input).

The output of your program is to indicate which round of the game the contestant is currently playing as well as the result of the game. There are three possible results:

```
You win.
You lose.
You chickened out.
```

Sample Input:

```
1
cheese
chese
2
cheese
abcdefg
3
cheese
abcdefgij
-1
```

Sample Output:

```
Round 1
You win.
Round 2
You chickened out.
Round 3
You lose.
```

Notre Dame ACM Programming Contest

Saturday April 8, 1995

Problem 6
"Rotating Sentences"

Description:

In "Rotating Sentences," you are asked to rotate a series of input sentences 90 degrees clockwise. So instead of displaying the input sentences from left to right and top to bottom, your program will display them from top to bottom and right to left.

As input to your program, you will be given a maximum of 100 sentences, each not exceeding 100 characters long. Legal characters include: newline, space, any punctuation characters, digits, and lower case or upper case English letters. NOTE: Tabs are not legal characters.

The output of the program should have the last sentence printed out vertically in the leftmost column; the first sentence of the input would subsequently end up at the rightmost column.

NOTE: Do not have trailing spaces at the end of any *line* (not column). In other words, the last character before the newline cannot be a space.

Sample inputs and outputs are on the next page.

Sample Input:

```
Rene Decartes once said,
“I think, therefore I am.”
```

Sample Output:

```
“R
Ie
 n
te
h
iD
ne
kc
,a
 r
tt
he
es
r
eo
fn
oc
re
e
 s
Ia
 i
ad
m,
.
”
```

Problem 7
“Tile Topology”

Description:

Your boss is re-tiling his bathroom and wishes to know how many unique ways there are to arrange an arbitrary number of tiles on the wall. He has assigned you to figure this out, but won’t tell you how many tiles he has. Design a program that accepts the number of tiles and returns the number of unique different arrangements. An arrangement is unique if it cannot be rotated in two dimensions to match any other arrangement.

For example, if there were 3 tiles, there are 2 possible arrangements:

```
❑❑❑      ❑❑
         ❑
```

Note that the following arrangement does not count because it can be rotated on a plane into one of the previous shapes:

```
❑
❑❑
```

Also note that the following two arrangements are different because they cannot be rotated into the same shape:

```
❑❑❑      ❑
❑        ❑❑❑
```

Sample Input:

```
3
```

Sample Output:

```
2
```

Notre Dame ACM Programming Contest

Saturday April 8, 1995

Welcome to the Notre Dame ACM Programming Contest.

There are 7 problems in this contest. There should be 11 pages (including this one) in your handout. You have 3 hours to solve as many as you can.

The problems are **not** arranged in any particular order of difficulty. It may be to your advantage to read through all the problems before you begin programming. Remember, it is the number of problems solved which counts.

Input to the programs (if any) is given through the standard input. That is, you may use `scanf(), gets(), cin,` or whatever. You will **not** have to open up any files to read input. Similarly, output should be sent to the standard output using `printf(), cout,` or whatever.

Answers must **exactly** match the judges' solutions. For example, a single misplaced comma, period, space, or newline will result in an incorrect submission. To make life a little easier, the output is fairly simple. For example, there are never any spaces at the end of a line. The output that we expect is in the same form as the sample output given in the problem descriptions.

Have fun!