2025 CCSC Eastern Conference Programming Competition

October 25th, 2025

Arcadia University, Glenside, PA

This page is intentionally left blank.

Problem 1 — Mirror Me

As an elite member of MI0, the legendary spy agency of Flufflandia (a small European country nestled between Luxembourg and Moldova), your job requires you to be proficient in all kinds of secret communication. The latest communication from chief spy "Y" indicates that on your next assignment you will need to be able to speak backwards, meaning you will speak each sentence as though all of the letters in the sentence occur in the opposite of their original order. In order to practice this arcane form of communication, you decide to write a program to take normal sentences and translate them to reversed form.

Program input: An "ordinary" line of input to the program will be a sequence of 0 or more characters followed by a newline character. As a special case, if the input line is "`quit`" followed by a newline, that means that the program input is done and the program should exit immediately.

Program output: For each ordinary line of input, the program should print one line of output (followed by a single newline), where the characters in the line are the reversal of the letters (A–Z/a–z) and space characters of the input line, with each letter converted to lower case. Characters other than spaces and letters should be omitted from the output.

Example input:

```
The secret plans are hidden in the burrito
My hovercraft is full of eels
A man, a plan, a cat, a canal - Panama!
quit
```

Example output (corresponding to the input shown above):

```
otirrub eht ni neddih era snalp terces eht
slee fo lluf si tfarcrevoh ym
amanap  lanac a tac a nalp a nam a
```

This page is intentionally left blank.

Problem 2 — Cooking with Fractions

Julia has started a recipe blog to share her amazing baking recipes with her followers. She has the recipes all ready and has written novella-length introductions for each recipe for her readers to scroll past. The only thing she needs help with is a feature that lets her readers multiply the recipe ingredients. This way if they want to double or triple the yield of the recipe, they can do so automatically.

You should help Julia by writing a program which will read in the amount to multiply the recipe by and a list of ingredients, and output the revised ingredient list with larger amounts. The amount to multiply will always be an integer and always be greater than one (so the amounts will only ever be increased, not decreased).

The amounts given in the ingredient list can appear as integers, fractions, or "mixed numbers" containing both a whole number and a fraction component. For example, the following are all valid ingredient amounts:

```
2 cups potatoes, peeled and chopped
3/4 teaspoon cloves
1 1/2 jars of marinara sauce
```

Your program should output the multiplied ingredient list using the same formats and in the simplest terms possible. For instance, if we were to double the ingredient list above, we should get the following:

```
4 cups potatoes, peeled and chopped
1 1/2 teaspoons cloves
3 jars of marinara sauce
```

Notice that the fraction `3/4` became a mixed number when doubled and the mixed number `1 1/2` became a whole number. Your program should never list a fraction which can be simplified, so for example `2/4` should be written as `1/2` instead. Likewise, you should not list something as a fraction which could be written as a whole number instead, so `4/2` should just be written as `2`. Finally, no fraction should have a numerator which exceeds its denominator, so `8/3` should be written as the mixed number `2 2/3` instead.

We may also need to add pluralization to an ingredient list. The rule for this is that all ingredients that are 1 unit or less should not be pluralized, while those of greater than 1 unit should be. Notice that `3/4 teaspoon` is not pluralized, while `1 1/2 teaspoons` is. To pluralize an ingredient, we can simply add an 's' to the first word after the amount. You can assume that the input will use correct pluralization.

Program input: The first line of input contains two integers, K and N, separated by a space. K is the amount we are multiplying each ingredient by, and will be an integer greater than 1. N is the number of ingredients in the recipe. Following this, there will be N lines of input, each giving one ingredient, in the format described above.

Program output: Output should consist of N lines, one for each of the ingredients in the input. Each should have its amount multiplied by K.

Example input:

```
3 12
1 cup butter, softened
3/4 cup white sugar
1 cup packed brown sugar
2 eggs
2 teaspoons vanilla extract
1 1/2 teaspoons baking soda
1/4 teaspoon lemon juice
1/2 teaspoon salt
1/3 teaspoon hot water
3 cups all-purpose flour
2 1/4 cups semisweet chocolate chips
1 1/3 cups chopped walnuts
```

Example output (corresponding to the input shown above):

```
3 cups butter, softened
2 1/4 cups white sugar
3 cups packed brown sugar
6 eggs
6 teaspoons vanilla extract
4 1/2 teaspoons baking soda
3/4 teaspoon lemon juice
1 1/2 teaspoons salt
1 teaspoon hot water
9 cups all-purpose flour
6 3/4 cups semisweet chocolate chips
4 cups chopped walnuts
```

Problem 3 — I Don't Even Care!

A truth table describes a boolean function by describing, for each combination of boolean input values, whether the output value should be true or false. Figure 3.1 below shows the truth table for the boolean function $q = a$ and (not b).

Certain boolean functions may be easier to specify if we allow "don't care" values for one or more inputs. A "don't care" value, denoted "X", means that it doesn't matter whether the input is true or false. Figure 3.2 below shows a truth table for a *selector* function. If the *en* input is F, the output q is false. Otherwise, the *sel* input determines whether the input *d1* or *d2* is copied to the output q: if *sel* is T, *d1* is copied to the output q, and if *sel* is F, *d2* is copied to the output q.

a	*b*	*q*
T	T	F
T	F	T
F	T	F
F	F	F

Fig. 3.1

en	*sel*	*d1*	*d2*	*q*
T	F	T	X	T
T	F	F	X	F
T	T	X	T	T
T	T	X	F	F
F	X	X	X	F

Fig. 3.2

Your task is to write a program to check boolean expressions to determine whether they correctly implement a function described by a truth table.

Program input: The input is a sequence of *functions*. Each function begins with a line with two integers, specifying the number of input variables and the number of truth table rows. As a special case, if the first line of a function consists of the single integer -1, that means that the input has ended and the program should exit immediately. Following the first line is a line with a comma-separated sequence of names of input variables. A variable name is a letter followed by 0 or more additional letters or digits (in any order.) Following the variable names is a sequence of truth table rows, each on its own line. The last column specifies the output value for each truth table tow. The other columns specify the input values, in the order in which the input variables were listed. After the truth table is a line with a single integer, denoting a number of *expressions*. Finally, there is a sequence of expressions, one per line. Each expression consists of a sequence of *tokens*. The tokens are left parenthesis, right parenthesis, `&` (and), `|` (or), `!` (not), and identifier. An identifier is a variable name (letter followed by 0 or more letters or digits.) Any space characters in an expression are ignored. As an example, the boolean function $q = a$ and (not b) could be written as the expression `a & !b`. `!` is the highest-precedence operator, and `|` is the lowest-precedence operator. Parentheses may be used to explicitly specify the order of operations. For example, in the expression `(a | b) & c`, the `|` operator is applied before the `&` operator.

Program output: For each expression, the program should output a single line of text reading either `yes` or `no`, indicating whether or not the expression was consistent with its truth table. An expression is not consistent with the truth table if there is at least one row of the truth table such that an assignment of truth values to input variables permitted by the truth table row causes the expression to yield an incorrect result value.

Example input:

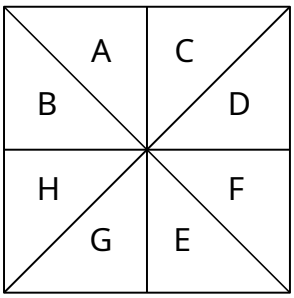
```
4 5
en,sel,d1,d2
T F T X T
T F F X F
T T X T T
T T X F F
F X X X F
3
en & ((!sel & d1) | (sel & d2))
(en & !sel & d1) | (en & sel & d2)
(!sel & d1) | (en & sel & d2)
5 6
en,sel1,d1,sel2,d2
F X X X X F
T T T X X T
T T F X X F
T F X T T T
T F X T F F
T F X F X F
4
en & ((sel1 & d1) | (sel2 & d2))
en & ((sel1 & d1) | (!sel1 & sel2 & d2))
(sel1 & en & d1) | (!sel1 & sel2 & d2)
(sel1 & en & d1) | (!sel1 & sel2 & d2 & en)
-1
```

Example output (corresponding to the input shown above):

```
yes
yes
no
no
yes
no
yes
```

Problem 4 — Kaleidoscope

Consider a square "image" consisting of N lines of text, where each line of text has N characters. We can transform this image into a "kaleidoscope" image by dividing the image into octants and then replicating the chosen octant and applying mirror symmetry to fill in the other octants of the image. Specifically, consider the image as having the following octants:



The generated image should copy the characters in the octant labeled "A" to all of the other octants (B–H), applying appropriate mirror symmetry. For example, all of the characters in octant B are mirrored from A across the diagonal between A and B.

As a special case, if width/height of the image is odd, meaning that the quadrants of the image can't be exactly the same size, the transformation should round up the width and height to the next larger even value, and "pretend" that the original image had those dimensions, but produce a result image that is the correct size by leaving out the right column and bottom row of characters.

Program input: The input is a sequence of *pictures*. Each picture is a single line containing a non-negative integer N, followed by a N lines of text. Each line of text will have exactly N characters. As a special case, if $N = -1$, then the input has ended and the program should exit immediately.

Program output: For each input picture with N lines, the program should output the "kaleidoscope" form of the picture as a sequence of N lines.

Example input:

```
6
......
.\O/..
..|...
..|...
./.\..
......
7
```

```
..^....
.<*>...
..v....
.....^.
....<*>
.|...v.
-*-....
-1
```

Example output (corresponding to the input shown above):

```
......
.\OO\.
.O||O.
.O||O.
.\OO\.
......
..^..^.
.<*>>*<
^*v..v*
.>....>
.>....>
^*v..v*
.<*>>*<
```

Problem 5 — Poker Calculator

Given a hand of five cards from a standard 52-card deck, calculate the probability that the hand will be better than all the hands of a given a number of opponents. Assume that there are no wild cards and that each hand is simply five cards that are drawn from a deck with no replacement. Assume that each hand is drawn from a separate deck, so you do not need to calculate the opponent's possible hand conditioned on your own hand.

A standard 52-card deck has 52 cards, where each card has a rank and a suit. There are 13 ranks: two through ten, jack, queen, king, and ace. Two is the lowest rank, and ace is the highest rank. There are 4 suits: spades, hearts, clubs, diamonds. All combinations of rank and suit are present in the deck. A poker hand consists of five cards. These are the types of poker hands, given from best to worst:

- Straight flush (SF): The five cards are all in the same suit and in consecutive ranks. An ace can be considered either low (in a straight of Ace-Two-Three-Four-Five) or high (Ten-Jack-Queen-King-Ace), but not both at the same time.
- Four of a kind (4K): Four cards with the same rank, and one other card.
- Full house (FH): Three cards of the same rank, and two other cards of the same rank (different from the first rank).
- Flush (FL): Five cards of the same suit
- Straight (ST): Five cards in consecutive ranks. As above, an ace can be either high or low but not both.
- Three of a kind (3K): Three cards with the same rank, and two other cards with different ranks.
- Two pairs (2P): Two cards with the same rank, two other cards with the same rank (different from the first rank), and one other card with a different rank.
- One pair (1P): Two cards with the same rank, and three cards with different ranks.
- High card (HC): Five unmatched cards.

If both players have the same type of poker hand, then there are rules to break the ties in a normal poker game. However, for our purposes, we will consider two hands as equivalent if they are the same type of poker hand. For example, any two hands that are both three of a kind are considered equivalent.

Program input: The first line of input has an integer n ($0 < n < 10000$), and that is followed by n additional lines. Each line begins with a positive integer less than 100000 indicating the number of other players. After a space, there are two characters indicating the type of poker hand. The two character code for each type of poker hand is listed above: SF, 4K, FH, etc.

Program output: For each additional line, output the probability that the poker hand indicated in that line is strictly better than all of the other players' randomly drawn poker

hands. Output your number as a percentage with two decimal places of precision.

Example input:

```
5
25 SF
1 1P
4 FL
50 ST
5000 4K
```

Example output (corresponding to the input shown above):

```
99.96
50.12
98.54
68.33
27.87
```

Problem 6 — PokéPairs: Type-Filtered Distinct Averages

In the world of competitive battling, a trainer's success often depends on balancing Pokémon of different types. Each Pokémon has an integer power level and a type, which may be Fire (F), Water (W), or Grass (G). You want to study the "balance" of power among Pokémon of certain types.

To measure balance, you perform the following ritual: For a given selection of types, collect all Pokémon of those types and sort their power levels. Then, repeatedly remove the current weakest and strongest Pokémon and record the average of their powers. Continue until no Pokémon remain. Your goal is to determine how many distinct averages appear during this process.

You will need to answer several queries, each specifying a subset of types. For each query, report the number of distinct averages for that subset.

Program input:

- The first line contains an integer n $(2 \leq n \leq 10^5)$, the number of Pokémon.
- The next n lines each contain an integer p_i $(1 \leq p_i \leq 10^9)$ and a character $t_i \in \{F, W, G\}$ — the power and type of the ith Pokémon.
- The next line contains an integer q $(1 \leq q \leq 10^5)$, the number of queries.
- The next q lines each contain a non-empty string of distinct letters from $\{F, W, G\}$, representing the types to include in that query (for example, "F", "FW", or "FWG").

Program output: For each query, print one line with a single integer — the number of distinct averages obtained for that subset of types.

Example input:

```
8
12 F
5 F
20 W
7 W
9 G
11 G
14 F
6 W
5
F
FW
FG
FWG
G
```

Example output (corresponding to the input shown above):

```
1
3
2
3
1
```

Problem 7 — Vampire Sort

It's Halloween night, and Count Vlad is organizing his vampire minions for the annual "Spooky Scavenge." He's lining them up, but he's a chaotic leader and has his own... unique method for organizing them. He calls it the Vampire Sort. The sort works like this: Vlad makes a single pass down the line of vampires, comparing two adjacent minions at a time. Each vampire has a "vitality" score (an integer). When he looks at a pair, the vampire on the left drains vitality from the vampire on the right. This "drain" (let's call it B) is mischievous. • For the very first comparison, B is 1. • For every comparison after that, the drain gets stronger, increasing by 13 from its previous value. After the drain happens (the left vampire's vitality goes up by B, and the right's goes down by B), Vlad compares their new vitality scores. If the left vampire now has more vitality than the right one, he forces them to swap places. The vampires and their new vitality scores then stay in place for the next comparison.

Example: Let's say three vampires are in line with vitalities: [10, 2, 57] 1. First Comparison (10 vs 2): o The drain B starts at 1. o Left vampire: 10+1=11 o Right vampire: 2-1=1 o Compare: Is 11>1? Yes. They swap. o The line is now: [1, 11, 57] (Note how their vitality scores have permanently changed). 2. Second Comparison (11 vs 57): o The drain B increases by 13. B is now 1+13=14. o Left vampire: 11+14=25 o Right vampire: 57-14=43 o Compare: Is 25>43? No. They do not swap. o The line is now: [1, 25, 43] 3. End of Pass: o The single pass is complete. The final vitality line-up is 1 25 43. Your Task Given the initial list of vampire vitalities, please determine the final list of vitalities after Count Vlad performs his single pass of the Vampire Sort.

Program input:

- The first line will contain T, the number of test samples, each of which takes two lines.
- The first line of each test sample will contain a single integer, N, representing the number of vampires in line.
- The second line will contain N space-separated integers, v1,v2,...,vN, representing the initial vitality of each vampire from left to right.
- Constraints: 2 <= N <= 35, -1,000 <= vi <= 1,000

Program output: Print a single line containing N space-separated integers, representing the final vitality scores of the vampires after the sort for each sample.

Example input:

```
2
3
10 2 57
5
7 -1 23 -8 42
```

Example output (corresponding to the input shown above):

```
1 25 43
-2 9 -35 2 89
```

Problem 8 — Scavenger Hunt

Joe likes to create scavenger hunts in his backyard to keep his children busy during the summer. He has created an 8x8 grid of land and buried something in each of the 64 cells. In some of the cells, he's buried prizes for the kids, such as a bag of candy, or the day's WiFi password.

In the rest of the cells he's buried a scrap of paper with a 2-digit number written on it. This number is a clue directing the kids to the next section of land to search. The tens digit indicates the row number and the ones digit indicates the column number. For example, 37 indicates row 3 and column 7. The rows and columns are both numbered 1–8.

His kids always begin the search in the upper-left hand corner of the grid, which is row 1, column 1. They then follow the clues until they either find a prize or are led back to a section of land they have already searched.

Your goal is to help Joe plan these hunts by writing a program which reads in what is buried in each section of land, and then simulating the search his kids will perform. At the end, you will know if the search leads to a prize or not.

Program input: The first line of input contains an integer N giving the number of test cases in the input. Following that are N test cases. There are no blank lines separating test cases. Each test case consists of 8 lines, giving the 8 rows of Joe's yard.

Each row contains 8 sets of 2 characters, separated by spaces, which indicate what is buried in each of the 8 columns in the row. These two characters are either two digits, which indicate a clue to another location, or "??" which indicates a prize is buried in this cell. You can assume that no clue will refer to a non-existent cell (such as 40 or 93).

Program output: For each test case, you should output the sequence of cells that the kids will search, represented as two-digit numbers, each on their own line. The first line of output should always contain "11" as that's where the search begins. The sequence ends with the first cell that either contains a prize or has already been visited. The last line of output for each test case should either be "Found a prize!" or "No prize found."

Example input:

```
2
87 55 86 88 76 48 61 43
84 73 17 52 73 27 66 43
33 12 84 21 81 31 72 11
57 32 21 14 76 33 83 24
73 37 58 55 75 37 14 68
76 24 54 38 56 21 33 62
48 76 45 83 75 ?? 21 72
27 46 15 22 83 24 51 81
17 15 73 14 24 13 86 33
81 54 ?? 82 37 56 21 85
11 75 32 86 83 53 16 18
82 75 37 83 44 45 ?? 23
78 11 13 18 68 82 32 53
37 78 86 16 86 45 43 22
21 84 27 82 ?? 72 23 31
85 86 86 22 47 46 12 87
```

Example output (corresponding to the input shown above):

```
11
87
51
73
45
76
Found a prize!
11
17
86
46
45
44
83
86
No prize found.
```