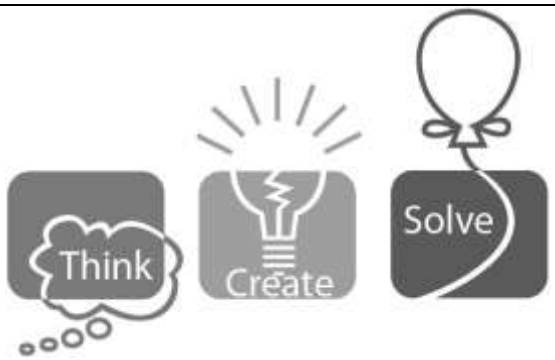
University of Central Florida



2022 (Fall) "Practice" Local Programming Contest

Problems			
Problem#	Difficulty Level	Filename	Problem Name
1	Easy	majestic	Majestic 10
2	Easy	palind	Phoneme Palindromes
3	Easy-Medium	breakice	Don't Break the Ice
4	Easy-Medium	search	Loopy Word Search
5	Medium	dreams	Wildest Dreams
6	Medium	findt	Dot the i's and Cross the T's
7	Medium-Hard	divide	Count the Dividing Pairs
8	Medium-Hard	jedi	Jedi and the Galactic Empire

Call your program file: filename.c, filename.cpp, filename.java, or filename.py

For example, if you are solving Phoneme Palindromes:

Call your program file: palind.c, palind.cpp, palind.java, or palind.py

UCF "Practice" Local Contest — Aug 27, 2022

Majestic 10

filename: `majestic`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

The movie "Magnificent 7" has become a western classic. Well, some years we have as many as 10 coaches training the UCF programming teams and once you meet them, you'll realize why they are called the "Majestic 10"! The number 10 is actually special in many different ways. For example, in basketball, they keep track of various statistics (points scored, rebounds, etc.) and if a player has 10+ (10 or more) in a particular stat, they call it a *double*.

The Problem:

Given three stats for a basketball player, you are to determine how many *doubles* the player has, i.e., how many of the stats are greater than or equal to 10.

The Input:

There is one input line, it contains three integers (separated by a space and each between 0 and 100, inclusive), providing the three stats for a player.

The Output:

Print a message indicating how many stats are greater than or equal to 10:

print `zilch` if none of the three stats is greater than or equal to 10,
print `double` if one of the three stats is greater than or equal to 10,
print `double-double` if two of the three stats are greater than or equal to 10,
print `triple-double` if all three stats are greater than or equal to 10.

Sample Input	**Sample Output**
`5 0 8`	`zilch`
`30 10 50`	`triple-double`
`20 5 20`	`double-double`
`5 100 6`	`double`

UCF "Practice" Local Contest — Aug 27, 2022

Phoneme Palindromes

filename: `palind`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

A *palindrome* is a string that reads the same forward and backward, e.g., `madam` and `abba`. Since some letters sound the same (e.g., c and k), we define a *phoneme palindrome* as a string that sounds the same forward and backward, e.g., `cak` and `ckckbbkcck`.

The Problem:

Given the letters that sound the same and a string, you are to determine if the string is a phoneme palindrome.

The Input:

The first input line contains an integer, p ($1 \le p \le 13$), indicating the count for pairs of letters that sound the same. Each of the following p input lines provides two distinct lowercase letters (starting in column 1 and separated by a space) that sound the same. Assume that no letter appears in more than one pair. The next input line contains an integer, q *($1 \le q \le 100$)*, indicating the number of strings to test for phoneme palindrome. Each of the following q input lines provides a string (starting in column 1 and lowercase letters only) of length 1 to 50, inclusive.

The Output:

For each string to test for phoneme palindrome, print a message (`YES` or `NO`) indicating whether or not the string is a phoneme palindrome.

(Sample Input/Output on the next page)

Sample Input	Sample Output
1 c k 6 a cac ck cab kaak ckckkcck	YES YES YES NO YES YES
2 a z x s 5 abbbz asxz cx sxxabzxss ks	YES YES NO YES NO

UCF "Practice" Local Contest — Aug 27, 2022

Don't Break the Ice

filename: `breakice`
Difficulty Level: `Easy-Medium`
Time Limit: `5 seconds`

Mr. Pennybags is enthralled with his new board game which involves knocking out tiny plastic ice blocks from a square grid. Pennybags was never very good at games and would like to practice his breaking strategies without having to reset the board. He has asked Unified Coders For Games (UCF Games) to develop such a tool. Pennybags wants a program to take in the description of a board and a sequence of moves and determine the number of invalid moves.

The Problem:

Given the dimensions (number of rows and columns) of a square game board and a list of moves, determine the number of attempted moves that would knock out an ice block that is no longer in the board. Note that when an ice block is knocked out, other blocks may fall out of the board as well. More specifically, an ice block will fall unless it is in a complete row (the row contains all its ice blocks) or it is in a complete column (the column contains all its ice blocks). Note also that if the fall of block B_1 results in the fall of block B_2, then other blocks may fall as a result of block B_2 falling.

The Input:

The first input line for the game contains two integers (separated by a space): the dimensions (length and width) of a square game board (between 1 and 50 inclusive) and the number of attempted moves in Pennybags' strategy (between 1 and 100 inclusive). This is followed by the row and column (separated by a space) for each attempted move, one move per line. Assume that the input *values* are valid, e.g., the row/column for an attempted move will always be a cell of the game board.

The Output:

Print the number of invalid moves for the game.

Notes:

1. An *invalid move* is defined as follows: selecting a cell that has either already been selected or the block at that cell has already fallen out.
2. The board is horizontal (in a plane parallel to the ground), i.e., the board is not vertical. So, when a block is fallen, there is no block above it to replace it.

Sample Input	Sample Output
4 5 1 1 1 2 4 1 4 2 1 1	2
4 4 1 3 2 4 1 4 4 4	1
3 3 1 1 2 2 3 3	0

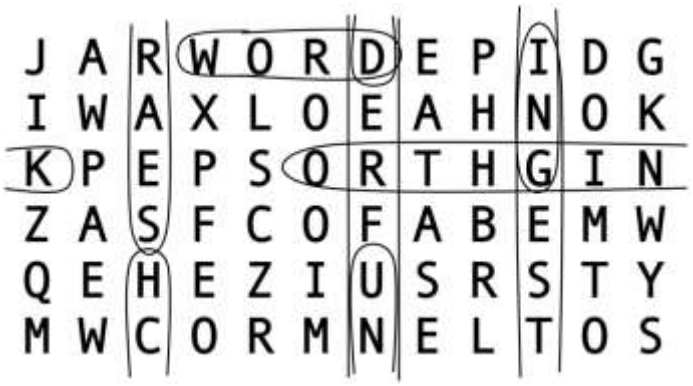
Explanation of the Sample Input/Output:

In the first Sample Input/Output, “4 2” and the second “1 1” are invalid.
In the second Sample Input/Output, “1 4” is invalid.

UCF "Practice" Local Contest — Aug 27, 2022

Loopy Word Search

filename: `search`
Difficulty Level: `Easy-Medium`
Time Limit: `5 seconds`



A word search puzzle is a grid of letters where your challenge is to find selected words as formed by consecutive letters in a line along the rows, columns, or diagonals of the grid. Tougher word searches also allow words in the grid to be forwards or backwards in any of those directions. In the "loopy word search", we will also allow words to go off the edge of the grid and continue (along the same line) on the other side, and potentially even reuse letters from that same word. However, in this problem, we won't search for words along diagonal lines, i.e., we only search along the rows and columns. (The UCF programming coaches are sure nice!)

The Problem:

Given a grid of letters and a list of words, identify the location of the first letter of each word in the grid and the direction in which remaining letters of the word can be found in sequence.

The Input:

The first input line for the puzzle contains two positive integers (separated by a space): *r*, the number of rows in the grid (between 3 and 12 inclusive), and *c*, the number of columns in the grid (between 3 and 20 inclusive). Each of the next *r* input lines for the puzzle contains exactly *c* uppercase letters, with no spaces. The next input line for the puzzle contains a positive integer *s*, the number of words to search for. Each of the next *s* input lines contains a string of uppercase letters (length between 3 and 100 letters, inclusive) which is a word to search for. It is not necessarily a real word in any language.

Each of the *s* words will appear exactly once in the grid, meaning it has exactly one starting location and goes only in one direction. None of the words will be palindromes (same letters backwards and forwards). Assume that the input is valid as described here.

The Output:

For each word given in the puzzle (and in the order given), output a line of the form "*d r c*" where *r* is the row in the grid where the first letter of the word is located (counting from 1), *c* is the column in the grid where the first letter is located (counting from 1), and *d* is the direction where the remaining letters of the word can be found, relative to the first letter, as given below. Output exactly one space after each of *d* and *r*. For the direction *d*, use the following 1-letter codes:

Code		Use for words with letters:
"R"	➔	in the same row that go to the **right**, into subsequent columns, potentially wrapping to the first column of the same row
"D"	➓	in the same column that go **down**, into subsequent rows, potentially wrapping to the first row of the same column
"L"	➔	in the same row that go **left**, back into previous columns, potentially wrapping to the last column of the same row
"U"	➑	in the same column that go **up**, into previous rows, potentially wrapping to the last row of the same column

(Sample Input/Output on the next page)

Sample Input	Sample Output
6 12 JARWORDEPIDG IWAXLOEAHNOK KPEPSORTHGIN ZASFCOFABEMW QEHEZIUSRSTY MWCORMNELTOS 5 WORD SEARCH KNIGHTRO UNDERFUND INGESTING	R 1 4 U 4 3 L 3 1 D 5 7 D 1 10
3 7 UCFAEHT KNIGHTS CODETRY 2 AGE THETHETHETHETH	D 1 4 U 3 5

UCF "Practice" Local Contest — Aug 27, 2022

Wildest Dreams

filename: `dreams`
Difficulty Level: `Medium`
Time Limit: `1 second`

Secretly, Arup loves Taylor Swift's music. He covers for his addiction by claiming that Anya, his nine-year-old daughter, forces him to play the Taylor Swift CD in the car whenever she's in it. The reality is that Arup also listens to the CD even when Anya isn't in the car.

Both of them have peculiar listening habits. Anya typically obsesses over a single song and will request that Arup play that song on infinite repeat while she is in the car. For example, the current song Anya is obsessed with is track 9, Wildest Dreams. So, when Anya enters the car, Arup immediately starts playing track 9 (from the beginning of the track) and this song is played repeatedly. Note that if Anya gets in the car while her favorite song is playing, the track will restart from the beginning of the song.

Arup, however, wants to listen to the whole CD and not just one track repeatedly. So, whenever Anya exits the car, Arup just lets the CD play in its natural ordering (tracks play in order and when the end of the CD is reached, track 1 starts again). If Anya exits the car in the middle of her favorite song playing, Arup just lets it continue to play that track until its end and then advances to the next song. For our example, if Anya exits the car in the middle of track 9, Arup lets track 9 continue until its end and then advance to the next song (track 10 or if track 9 is the end of CD, track 1).

The Problem:

Arup is curious exactly how long he has listened to Anya's favorite song. Given a list of the lengths of each song, when Anya is in the car and the song Anya is obsessed with, calculate the amount of time Arup has to hear the song that Anya is obsessed with. Recall that when Anya enters the car, her song is played from the beginning and the song keeps repeating as long as Anya is in the car.

You may assume that if Anya gets out of the car in the middle of Anya's song playing, Arup will continue listening to it instead of forwarding the CD to the next song but when Anya's song is finished the next song will continue. If Anya's song ends exactly when Anya is leaving the car, the next song will continue, i.e., Anya's song does not loop back to the beginning.

The Input:

The first line of input for the CD will contain two integers: t ($1 \le t \le 20$), the number of tracks on the CD, and k ($1 \le k \le t$), the track that Anya is obsessed with. The second line of input for the CD contains t space separated positive integers representing the lengths of each of the t tracks on the CD, in order, in seconds. No CD will be more than 86,400 seconds in length (the number of seconds in a day).

The third line of input for the CD contains a single positive integer, d ($1 \le d \le 100$), the number of days to evaluate. The following d lines contain information about each day. Each of these lines will start with an integer, s_i ($1 \le s_i \le 20$), the number of segments of driving for day i. This is followed by s_i positive integers, representing the lengths of each of the driving segments, in seconds. Assume that Anya is in the car for the odd-number segments (first, third, fifth, etc.). The sum of the lengths of each driving segment will never exceed 86,400 (the number of seconds in a day).

The Output:

The output will consist of d lines, where d represents the number of days that the CD was played. Output the number of seconds Arup listed to Anya's favorite song on the CD, for each day, respectively.

Sample Input	**Sample Output**
13 9 212 231 231 235 193 219 207 211 220 247 250 195 270 4 3 1000 900 1000 3 10000 10000 10000 1 2000 2 500 600	2100 20780 2000 660
2 2 100 200 5 1 70 5 300 277 131 10000 58 2 200 50 2 201 50 2 199 50	70 7335 200 251 200

UCF "Practice" Local Contest — Aug 27, 2022

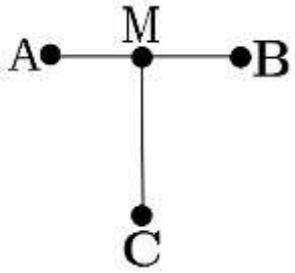
Dot the i's and Cross the T's

filename: `findt`
Difficulty Level: `Medium`
Time Limit: `1 second`

Mr. T, known for his hair cut and the phrase "you fool" in the old TV series "The A Team", has decided to try out for the UCF Programming Team. Considering the many talented students at UCF, Mr. T's best chance is to become great at geometry. So, he sought help from Euclid, one of the Fathers of Geometry. Considering the teacher and the student, the obvious place to start is with the letter T!

Consider the picture of the letter T to the right.
We define four points A, M, B, and C form a T if three conditions hold:

1) M is the midpoint of AB
2) CM = AB, i.e., CM is the same length as AB
3) The angles AMC and BMC are 90 degrees.



The Problem:

Given a set of points, you are to determine how many groups of four points form a T based on the above definition.

The Input:

The first input line contains an integer, p ($4 \le p \le 50$), indicating the number of points. Each of the following p input lines provides two real numbers (each between −1000 and 1000, inclusive), indicating (respectively) the x and y coordinates for a point. Assume all points are distinct. Also assume that the x and y coordinates in the input will have at most three digits after the decimal point. This will limit the accumulation of partial errors while performing the intermediate operations.

The Output:

For the given set of points, print how many groups of four points form a T. Two groups of four points are considered different if they differ in at least one point.

Note/Hint:

This problem deals with floating-point numbers and one must be careful about checking for equality of two values. Assume two values are equal if they differ by 10^{-6} or less.

Sample Input	Sample Output
5 5.0 2.0 3.0 6.0 5.0 6.0 7.0 6.0 5.0 10.0	2
4 -3.0 6.0 0.0 6.0 3.0 6.0 0.0 12.0	1

UCF "Practice" Local Contest — Aug 27, 2022

Count the Dividing Pairs

filename: `divide`
Difficulty Level: `Medium-Hard`
Time Limit: `2 seconds`

Number Theory provides many fascinating properties. You have most likely written programs dealing with different groups of numbers such as Prime, Perfect, Amicable, Happy, Powerful, and Untouchable numbers, just to name a few. In this problem, you'll attack yet another fascinating property of numbers, one dealing with pairs of numbers.

An integer D is said to be a proper divisor of an integer N if $D \neq N$ and there exist an integer Q such that $N = Q * D$. For example, 4 is a proper divisor of 8 and 5 is a proper divisor of 15, but 9 is not a proper divisor of 9 and 6 is not a proper divisor of 8. Note that zero is not a proper divisor of any number but all numbers (except zero) are proper divisors of zero.

We will call (D, N) as defined above "proper dividing pairs".

The Problem:

Given a list of integers $A = \{A_1, A_2, \ldots, A_p\}$, you are to determine (count) the number of proper dividing pairs (A_i, A_j), where $1 \leq i, j \leq p$.

The Input:

The first input line contains an integer, p ($2 \leq p \leq 10^6$), indicating the number of integers in the list. The following input line will provide p integers, A_i ($0 \leq A_i \leq 10^7$).

The Output:

For the input list of numbers, print the number of proper dividing pairs.

Note that, as illustrated in Sample Input/Output (next page), duplicate values in the input list are considered as different elements in the list and they each contribute to the total count (proper dividing pairs).

Sample Input	Sample Output
3 1 2 3	2
4 1 2 3 1	4
2 7 5	0
3 29 0 17	2
10 32 16 8 4 2 2 4 8 16 32	40

UCF "Practice" Local Contest — Aug 27, 2022

Jedi and the Galactic Empire

filename: `jedi`
Difficulty Level: `Medium-Hard`
Time Limit: `1 second`

Jedi Knights are often tasked with protection. Whether protecting shield generators or important diplomats, the Jedi will use their lightsabers to deflect blaster shots and keep their asset safe.

Sometimes a Jedi will go on missions alone or will be accompanied by another Jedi. When protecting an asset, they will stand side by side deflecting shots that would otherwise harm the asset they wish to protect. Sometimes, even together, they cannot block all the blaster shots. That is because Jedi are still limited by their physical reaction time. More specifically, when a Jedi blocks a blaster shot, he has to wait a certain amount of time before he can block another shot. For example, a Jedi that takes t time units between shots can block a shot arriving at time k and a shot arriving at time $k + t$ (or later).

So, Jedi will coordinate which shots they each will block and which shots they will allow to pass through their defense. Either Jedi can independently block each shot as long as there is enough time between the current shot and his last blocked shot. But determining which shots to block and which to let by is no easy task if they want to minimize the number of shots that reach their asset. That is why they seek aid from the other great power in the galaxy, programming.

As the Jedi's knowledge of programming is not as deep as their knowledge of the force, they have asked the programmers of the Universal Computational Federation (UCF) to find better strategies for blocking blaster shots with their lightsabers.

The Problem:

Given the times the blasters reach the Jedi, the number of Jedi, and their reaction time, determine the number of blaster shots that reach the asset they are trying to protect. Note that each Jedi can block a blaster shot at the beginning of the mission but after his first block the Jedi is limited by his reflexes (the time he has to wait before he can block another shot).

The Input:

The first input line contains an integer, b ($1 \le b \le 1{,}000$), the number of blaster shots fired at the Jedi's asset. This is followed by a line containing b numbers, separated by spaces, which are the times the blaster shots will reach the Jedi. These numbers can be in any order but will be positive integers less than or equal to 1,000. This is followed by an integer j ($1 \le j \le 2$) on a line by itself, which is the number of Jedi on the mission. The last input line contains j space separated integers giving the reaction time of each Jedi, the time it takes a Jedi to prepare to block the next blaster shot. These numbers will be between 1 and 100 inclusive.

The Output:

Output the minimum number of blaster shots that the Jedi are unable to block and will hit their asset.

Sample Input	Sample Output
5 10 5 5 10 5 1 100	4
4 2 4 9 9 2 10 7	1
5 2 4 8 13 13 2 10 7	1
5 2 4 6 8 10 1 2	0

Explanation of the Sample Input/Output:

In Sample Input #2, Jedi with speed 7 can block 2 and one 9 and Jedi with speed 10 can block 4, resulting in one shot remaining unblocked.
In Sample Input #3, Jedi with speed 7 can block 4 and one 13 and Jedi with speed 10 can block 2 and the other 13, resulting in one shot remaining unblocked.