Computer Science Competition
Invitational B 2022
Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Anastasia
Problem 2	Carolyn
Problem 3	Diana
Problem 4	Ewa
Problem 5	Harry
Problem 6	Ishita
Problem 7	Manoj
Problem 8	Micha
Problem 9	Pamela
Problem 10	Rhea
Problem 11	Shekhar
Problem 12	Tomas

1. Anastasia

Program Name: Anastasia.java **Input File: None**

Anastasia's little sister just learned about the square root of numbers in her elementary math class. Recall, that the square root of a number x is a number y such that $y^2 = x$. For example the square root of 400 is 20. This is true because $20^2 = 20 * 20 = 400$. Anastasia wants to make her little sister a study guide that can be used to check the square roots of the following numbers: 400, 361, 324, 289, 256, 225, 196, 169, 144, 121, 100, 81, 64, 49, 36, 25, 16, 9, 4, 1, and 0. Can you help her with this?

Input: There is no input for this problem.

Output: For each of the following numbers: 400, 361, 324, 289, 256, 225, 196, 169, 144, 121, 100, 81, 64, 49, 36, 25, 16, 9, 4, 1, and 0, you are to output "The square root of X is Y." Where $Y^2 = X$.

Sample output:

```
The square root of 400 is 20.
The square root of 361 is 19.
The square root of 324 is 18.
The square root of 289 is 17.
The square root of 256 is 16.
The square root of 225 is 15.
The square root of 196 is 14.
The square root of 169 is 13.
The square root of 144 is 12.
The square root of 121 is 11.
The square root of 100 is 10.
The square root of 81 is 9.
The square root of 64 is 8.
The square root of 49 is 7.
The square root of 36 is 6.
The square root of 25 is 5.
The square root of 16 is 4.
The square root of 9 is 3.
The square root of 4 is 2.
The square root of 1 is 1.
The square root of 0 is 0.
```

2. Carolyn

Program Name: Carolyn.java **Input File: carolyn.dat**

Carolyn enjoys the simple things in life: sunsets, old movies, and quiet walks are among her favorites. Also on her list are triangles and letters of the alphabet. She needs you to write a program to utilize those last two items.

Write a program that accepts the input of a two-character string. Create the right triangle by moving from the first letter to the second letter.

Example: `AC` would create the following triangle.

```
A
BB
CCC
```

Note: If the second letter occurs before the first letter in the alphabet, the order of the letters will be in reverse.
Example: `CA` would create the following triangle.

```
C
BB
AAA
```

Input: The first line consists of a number N, representing the number of lines of data to follow. N will be in the range of [1,50]. The next N lines of data consist of one string that is two characters in length. Each character will be an upper-case letter.

Output: Each output will consist of a right triangle of letters starting with the first letter and ending with the second letter. Each row in the right triangle will increase in length by one as you move from row-to-row.

Sample input:

```
5
AD
HH
PL
JS
DA
```

Sample output:

```
A
BB
CCC
DDDD
H
P
OO
NNN
MMMM
LLLLL
J
KK
LLL
MMMM
NNNNN
OOOOOO
PPPPPPP
QQQQQQQQ
RRRRRRRRR
SSSSSSSSSS
D
CC
BBB
AAAA
```

3. Diana

Program Name: Diana.java **Input File: diana.dat**

Diana recently completed an Algebra class where she learned about the Cartesian or X-Y coordinate system where points are identified by x-y coordinates as point *p* with coordinates (x, y). One of the things that really amazed her was the formula for computing the distance between any two points. Her teacher mentioned that the formula for a 3-dimension (3-D) coordinate system with an additional Z axis is similar. A 3-D point is defined by adding a third value, *z*, for the third dimension, so point *p* becomes (x, y, z).

Given two 3-D points p_1 as (x_1, y_1, z_1) and p_2 as (x_2, y_2, z_2): the distance between the points can be calculated using the following formula.

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Diana is still a little awkward with the calculator and has asked the school programming team to build her a program she can use to double-check her calculations.

Write a program to calculate the distance between any two 3-D points with distances rounded to exactly 2 decimal places as shown in the sample output.

Input: A list with no more than 20 sets of coordinates. Each line will contain two complete sets of floating point coordinates with six values separated by whitespace as shown below. The individual coordinate values (V) will be in the range $-50{,}000.00 \leq V \leq 50{,}000.00$. Each line of input contains values in the following order:

$$x_1 \quad y_1 \quad z_1 \quad x_2 \quad y_2 \quad z_2$$

Output: For each set of coordinates, output a single line containing the distances rounded to 2 decimal places.

Sample input:

```
-5359.76 32044.22 7882.16 45156.50 -41396.57 5497.94
-37525.56 9046.11 2285.20 -45759.01 13547.75 -3877.51
48809.73 4712.09 -28657.86 4186.49 26991.17 -33937.56
35847.36 -19489.78 -46732.41 44807.97 2227.51 28611.97
```

Sample output:

```
89169.09
11226.46
50154.42
78922.17
```

4. Ewa

Program Name: Ewa.java **Input File: ewa.dat**

Eva is at a family gathering. She loves eating pie, but hates having to go to the kitchen to check how much is left. In order to get around this, Eva wants to write a program that will track how much pie enters and leaves the kitchen. Help Eva write such a program.

Input:
The first line consists of a number T (1 <= T <= 20), representing the number of test cases that follow.
Each test case will begin with a single integer N, the number of events that occur in that test case.
The following N (1 <= N <= 20) lines of each test case will each consist of a single event in chronological order. There are two types of events:

- `BRING <Radius> <Depth>` denotes that some family member brought a new cylindrical pie into the kitchen of the given radius and depth.
- `EAT <Volume>` denotes that some family member took a slice of pie of the given volume from the kitchen.

Output: After each event, output the volume of pie remaining in the kitchen, rounded to 2 decimal places. Print a line containing 10 dashes "`----------`" between test cases.

Sample input:
```
2
4
BRING 1.0 1.0
BRING 2.0 2.0
EAT 3.14
BRING 1.5 1.5
1
BRING 1.0 1.0
```

Sample output:
```
3.14
28.27
25.13
35.74
----------
3.14
----------
```

5. Harry

Program Name: Harry.java **Input File: harry.dat**

Harry is always concerned that he will be in a situation where he needs to send a message written in code. Since he is someone who is always prepared, he has created a simple way to encode and decode a message.

His technique is to create two strings. The first is his secret message hidden by other letters. The second string consists of all the letters that should be removed from the first string in order to reveal the message.

For example: CANHDDDEYYLNACPAAMDDEDY CANDY

If you remove the letters from CANDY you get … HELPME

Let's help Harry. It is simply the right thing to do.

Write a program for Harry that will read in two strings - StringA and StringB. Remove all the letters from StringA that occur in StringB. Then, and only then, we will see Harry's secret message.

Input: The first line consists of a number N, representing the number of lines of data to follow. N will be in the range of [1,50].

The next N lines of data consist of two strings consisting of all uppercase letters. The strings are separated by one space. The first string is the encoded message. The second string consists of the letters which should be removed from the encoded message.

Note: If all letters have to be removed from the encoded message, your program should output the special message "`ALL LETTERS ARE GONE`"

Output: Each output will consist of either the decoded message - a string of length [1,255], or the message "`ALL LETTERS ARE GONE`"

Sample input:

```
6
NILBBIAKEABEGGBS BANANA
GRUNFORIRESTRPUN PIG
SUPERSTAR GOLD
ZBOEYKGGIYNLDZ ZOOLOGY
BTHOOEERAYLE BEETLE
TEXAS TEXAS
```

Sample output:

```
ILIKEEGGS
RUNFORRESTRUN
SUPERSTAR
BEKIND
HOORAY
ALL LETTERS ARE GONE
```

6. Ishita

Program Name: Ishita.java **Input File: ishita.dat**

Ishita has drawn out a long sequence of parenthesis. He would like to cut the sequence into pieces such that the parentheses on every piece are balanced. A sequence of parentheses is considered balanced if every '(' can be matched by a unique ')' that comes later in the sequence..

Given the sequence that Ishita has drawn out, determine how many ways there are to cut the sequence.

Input:
The first line consists of a number T (1 <= T <= 10), representing the number of test cases that follow.
Each test case will consist of a single line of up to 30 '(' and ')' characters describing the sequence Ishita drew.

Output: For each test case, output the number of ways there are to cut the sequence

Sample input:
```
3
()(())()
())(()
()()(((())))(())
```

Sample output:
```
3
0
7
```

Explanation for sample case 1:
Three ways to cut, pieces separated by period:
()(()).()
()(())()
()((())).()

7. Manoj

Program Name: Manoj.java **Input File: manoj.dat**

Manoj has always been interested in temperature scales. Of course he has studied a lot about the Celsius and Fahrenheit scales. He believes it is time to create a Manoj Scale. Your job is to create the Manoj Scale given the proper information, then allow him to convert to and from the three scales.

Here is some vital information:
On the Celsius Scale, the freezing point of water is 0 degrees and boiling point of water is 100 degrees.
On the Fahrenheit Scale, the freezing point of water is 32 degrees and boiling point of water is 212 degrees.

Your program will read the freezing and boiling points of water on the Manoj Scale.

Using those two numbers on each scale, you should be able to convert from any scale to any of the others.

To make this challenge more intriguing, Manoj is only going to use integer inputs, and beyond that the data he provides will only create integer results. This is because Manoj loves mental math challenges involving integers.

For this problem, C means Celsius, F means Fahrenheit, and M means Manoj.

Input: The first line consists of a number N, representing the number of lines of data to follow. N will be in the range of [1,50]. The next N lines of data consist of three integers Num1, Num2, and Num3 followed by two characters Char1 and Char2. A single space separates each of the five items.

Num1 will be the freezing point of water on the Manoj Scale. It will be an integer in the range [-1000,1000].
Num2 will be the boiling point of water on the Manoj Scale. It will be an integer in the range [-1000,1000].
Note that Num2 > Num1.
Num3 will be the temperature on the Char1 that is being investigated. It will be an integer in the range [-1000,1000].
Char1 will be an uppercase C, F, or M indicating the scale from which we are converting.
Char2 will be an uppercase C, F, or M indicating the scale to which we are converting.
Note that Char1 and Char2 will not be equal to each other.

Output: Each output will be a statement following the formatting rules below.
<Temperature to convert> degrees <starting scale> = <Converted value> degrees <destination scale>
One space separates each item in the output.

Sample input:
```
6
0 50 20 M C
100 190 145 M F
32 212 171 F M
30 70 77 F M
0 10 80 C M
10 20 95 C F
```

Sample output:
```
20 degrees M = 40 degrees C
145 degrees M = 122 degrees F
171 degrees F = 171 degrees M
77 degrees F = 40 degrees M
80 degrees C = 8 degrees M
95 degrees C = 203 degrees F
```

8. Micha

Program Name: Micha.java **Input File: micha.dat**

Micha has noticed that some letter sequences like “ing” and “tion” among many other combinations seem to appear frequently in commonly used words. Such patterns can be used as clues for deciphering encrypted messages. However, trying to find the most common patterns can be a real challenge as she visually scans sentences. She would like a program that can flawlessly complete the task and help her gather data.

Micha will provide the text to be scanned along with a list of suspected common letter sequences. She would like to get a report showing the number of times specific sequences were found in the text. To make your task easier, all punctuation marks including hyphens and special characters have been eliminated and replaced by spaces if needed. There could be numbers in the text.

Input: The first line is a positive integer $S \leq 15$, the number of letter sequence for which to scan. That will then be followed by an unknown amount of text on an unknown number of lines which is to be scanned for each of the desired letter sequences. The data file will not exceed 100 lines and individual line length will not exceed 100 characters. Letter case is not considered significant, “ing” and “ING” are the same sequence.

Output: For each letter sequence provided, output the sequence surrounded by double quotation marks followed by a single space and the count of occurrences for that sequence.

Sample input:

```
3
ing
tion
ern
Micha has noticed that some letter sequences like ing and tion among many
other combinations seem to appear frequently in commonly used words Such
patterns can be used as clues for deciphering encrypted messages However
trying to find the most common patterns can be a real challenge as she
visually scans sentences She would like a program that can flawlessly
complete the task and help her gather data
```

Sample output:

```
"ing" 3
"tion" 2
"ern" 2
```

9. Pamela

Program Name: Pamela.java **Input File: pamela.dat**

Pamela has recently started learning about 2-dimension arrays that contain rows and columns of data. Before she starts with more complex operations, she wants to practice with the basics. Her goal is to be able to consistently compute the sums of individual rows and columns and find minimum values for the columns and maximum values for the rows for arrays of various sizes. The following 5 x 4 sample shows the results she will calculate:

Columns ↘ Rows ↓	0	1	2	3	Row Sums	Row Mins
0	160	918	572	587	2237	160
1	817	155	703	903	2578	155
2	471	468	962	311	2212	311
3	890	575	532	128	2125	128
4	266	259	442	167	1134	167
Col Sums	2604	2375	3211	2096		
Col Maxs	890	918	962	903		

Can you produce the same result so Pamela can check hers?

Input: The first line is a positive integer $1 \le T \le 10$, the number of test cases in the data file. That will then be followed by T sets of data. For each dataset, the first line will contain 2 integers: the number of rows (R) and the number of columns (C) with $2 \le R, C \le 15$. The dataset continues with R rows, each containing C integers (N) to populate the individual array cells with $0 \le N \le 1000$.

Output: For each test case, the first line contains a case number, formatted as shown in sample. The next four lines contain the row sums, the row minimums, the column sums, and the column maximums. The next two lines contain the overall minimum and the overall maximum. All lines must be labeled and formatted as shown below with integers displayed in right-aligned fields that are 7 positions wide and no additional spacing. The final line for each test case will contain 20 equal signs "==========". There are no blank lines.

Sample input:

```
2
5 4
160   918   572   587
817   155   703   903
471   468   962   311
890   575   532   128
266   259   442   167
7 9
286   523   961   240   866   234   252   688   437
922   182   702   925   651   613   820   477   580
10    516   533   639   239   51    538   300   268
620   473   663   705   10    210   85    597   613
459   608   828   465   669   327   932   174   950
984   413   465   788   958   760   817   402   531
571   511   757   62    581   444   650   271   65
```

See next page for sample output…

Pamela continued…

Sample output:

```
Case: 1
Row Sums:   2237   2578   2212   2125   1134
Row Mins:    160    155    311    128    167
Col Sums:   2604   2375   3211   2096
Col Maxs:    890    918    962    903
Min Mins:    128
Max Maxs:    962
====================
Case: 2
Row Sums:   4487   5872   3094   3976   5412   6118   3912
Row Mins:    234    182     10     10    174    402     62
Col Sums:   3852   3226   4909   3824   3974   2639   4094   2909   3444
Col Maxs:    984    608    961    925    958    760    932    688    950
Min Mins:     10
Max Maxs:    984
====================
```

6. Rhea

Program Name: Rhea.java **Input File: rhea.dat**

Rhea is at the mall, where the floor tiles are many different colors. As a fun game, Rhea only wants to walk on tiles in order of the rainbow (Red, Orange, Yellow, Green, Blue, Indigo, Violet). If Rhea is on a certain color tile, she is only willing to move to an adjacent tile if it is the same color, or the next color in rainbow order (This order is not cyclical). Given this restriction, help Rhea determine if such a path exists from her current location to the smoothie shop.

Input:
The first line will consist of a single integer T (1 <= T <= 10), the number of test cases to follow.
Each test case begins with two integers L and W (1 <= L,W <= 50) on their own lines, denoting the length and width of the mall in floor tiles.
The next L lines will each consist of a row with W floor tiles.
Rhea begins on a red tile, and her location will be denoted by a '*'
The smoothie shop is on a purple tile, and its location will be denoted by a '#'.
Any other tiles will be represented with the first letter of their respective colors.

Output:
Output `yes` if it is possible for Rhea to reach the smoothie shop from her current location, and `no` otherwise.

Sample input:
```
3
5 5
*RRRR
VVVVO
YYYYY
GGGGG
BIVV#
2 5
*YGG#
OYVBI
1 5
*YGB#
```

Sample output:
```
yes
yes
no
```

11. Shekhar

Program Name: Shekhar.java **Input File: shekhar.dat**

Shekhar just got the new and improved Computron-2021 graphing calculator, made by Lone Star State Devices. The Computron-2021 can do all mathematical calculations from solving algebraic equations to taking derivatives of multivariable functions. Being the top of the line calculator it is, Shekhar is bound and determined to try and find a bug with the calculator so he can report back to Lone Star State Devices in hopes of earning an academic scholarship.

Shekar believes the Computron-2021 doesn't ensure that the delimiters of a mathematical expression are nested correctly. The Computron-2021 uses parentheses ( ), brackets [ ], and braces { } as delimiters. For a mathematical expression to have correctly nested delimiters, the following two statements must be true:

1. There are an equal number of right and left delimiters
2. Every right delimiter is preceded by a matching left delimiter.

For example, the expression $7 - ((x * ((x + y) / (j - 3)) + y) / (4 - 2.5))$ is nested correctly because the two above conditions are satisfied. The expression $((a + b)$ is not nested correctly because there are two left parentheses but only one right parenthesis, thus causing statement 1) above to be not met. The expression $\{a * (b + c\})$ is not nested correctly either. Although there are equal numbers of right and left delimiters, the right brace } is preceded by a left parenthesis (, thus causing statement 2) not to be met.

Shekar needs your help writing a program that can be used to double check all the challenging, weird, and wonky expressions he plans on using to break the Computron-2021.

Input: The input begins with an integer I, the number of test cases. I is guaranteed to be in range [1,30]. The following I lines will contain a single mathematical expression. The expression may constitute integer literals, variables, addition, subtraction, multiplication, and division, but the only delimiters allowed are parentheses, brackets, and braces.

Output: For each mathematical expression, you are to output the expression followed by either "`is nested correctly`" or "`is nested incorrectly`"

Sample input:

```
12
7 - ((x * ((x + y) / (j - 3)) + y) / (4 - 2.5))
((a + b)
{a*(b+c})
7 - ({x * [(x + y) / (j - 3)] + y} / (4 - 2.5))
a + b(
)a + b(
{[(a + b) * 10 + c] * 10 + d}
((((((((((()))))))))))
((((((((((()))))))))))(
)((((((((((()))))))))))
```

Sample output:

```
7 - ((x * ((x + y) / (j - 3)) + y) / (4 - 2.5)) is nested correctly
((a + b) is nested incorrectly
{a*(b+c}) is nested incorrectly
7 - ({x * [(x + y) / (j - 3)] + y} / (4 - 2.5)) is nested correctly
a + b( is nested incorrectly
)a + b( is nested incorrectly
{[(a + b) * 10 + c] * 10 + d} is nested correctly
((((((((((()))))))))))) is nested correctly
((((((((((()))))))))))) is nested incorrectly
((((((((((()))))))))))) is nested incorrectly
((((((((((()))))))))))(( is nested incorrectly
)((((((((((()))))))))))) is nested incorrectly
```

12. Tomas

Program Name: Tomas.java **Input File: tomas.dat**

Tomas was visiting with his parents, and they were telling him about text messaging in the late 20th century, and in the early 21st century. To Tomas's surprise, most cellular phones during that time did not have full keyboards! Instead, the devices only had a numeric keypad (1-9 and 0), *, and #. In order to type alphabetic characters, the below numeric keypad was used:

1	2 A B C	3 D E F
4 G H I	5 J K L	6 M N O
7 P Q R S	8 T U V	9 W X Y Z
*	0 –	#

For example, to type the character 'M', the device user would press the 6 button one time, to type the character 'Y', the user would press the 9 button three times, and to type a ' ' (space) the user would press the 0 button. Now, it was very common for users to inadvertently press a button too many times, in which case the list of available letters would cycle. For example, if the user pressed the 3 button four times, the typed letter would be 'D', and if the user pressed the 7 button eleven times, the letter would be 'R'.

Tomas would like your help writing a program that can be used to translate numeric keypad entries into alphabetic text. For example if the following input was given:
88 444444 555 0 444 7777 0 2222222 9999999999999 33333333333 77777777 666 6 33
The result would be: UIL IS AWESOME

A further breakdown of the input shows:
8 pressed twice results in a 'U'
4 pressed six times results in an 'I' (this is an example where the list of letters cycled)
5 pressed three times results in a 'L'
0 pressed one time results in a ' ' (space)
4 pressed three times results in an 'I'
7 pressed four times results in a 'S'
0 pressed one time results in a ' ' (space)
2 pressed seven times results in an 'A' (this is an example where the list of letters cycled)
9 pressed thirteen times results in a 'W' (this is an example where the list of letters cycled)
3 pressed eleven times results in an 'E' (this is an example where the list of letters cycled)
7 pressed eight times results in a 'S' (this is an example where the list of letters cycled)
6 pressed three times results in an 'O'
6 pressed one time results in a 'M'
3 pressed twice results in an 'E'

Can you help Tomas write such a program?

Continued next page…

Tomas continued…

Input: The input begins with an integer I, the number of test cases. I is guaranteed to be in range [1,20]. Each test case will be separated by a blank line to distinguish where one test case starts, and the next test case begins (There will be a blank line after I). Each non-blank line will contain the numeric keypad entries to be translated, a series of buttons pressed by the device user. The buttons given will be guaranteed to be 0 or 2 through 9 only. The buttons 1, *, and # will not be given as input, so no input validation is needed. Each grouping of key presses will be separated by a space. 0 represents a ‘ ‘ (space) and all other keypad entries follow that of the figure above. Don’t forget, cycles in the letters may be present, as shown in the above example.

Output: For each input case, you are to translate the numeric button presses to the corresponding alphabetic message. All alphabetic output should be upper case letters. Each line of output should be on its own line.

Sample input:

```
5

88 444444 555 0 444 7777 0 2222222 9999999999999 33333333333 77777777 666 6 33

6 999 0 777777777 2222 77777777777 33333 66666 8888 77777777 0 7777777 33333 2222
555555555 555 9999999 0 8 33 99 8888 33 3333333333 0 555555555555 444444444 55
33333333 0 8888888 44 444 7777

6666666666 2 999 22 33333 0 77777777777777 999999999 33333 777 8 9999999 0 55555555555
33333333 999 22222222 666666 2222 777 3333333333 7777 0 2222222222 777 33333 0 66 666
8888 0 7777777777777777 666 0 22222222222 2222222 3

8 44444 33 0 77 88888888888 444 222222 55555555555 0 22 777777777777777 666 9 66 0
333333333 666 999999 0 5555555555 88 6 777777777 7777777777777777 0 666 888 33333
77777777777 0 8 44 33333333 0 555555555555 2 99999999 999 0 3 666666666666 4444444

333333333 666666666 88 777 0 77777777 222222222 666 77777777777 33333 0 2 66 3 0 7777
33 888 33 66 0 999999999999999 33333 2 777 7777777777777777 0 2 4444 666666666
```

Sample output:

```
UIL IS AWESOME
MY PARENTS REALLY TEXTED LIKE THIS
MAYBE QWERTY KEYBOARDS ARE NOT SO BAD
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
FOUR SCORE AND SEVEN YEARS AGO
```