DKC^3 2020 – Long Programming Problems

Important Reminders:

- Time remaining will be announced at 30 min., 5 min. and 1 min. left in the session.

- Assume leading zeros are used in all decimal notations unless the problem says otherwise. (ex: 0.167)

- Example input and output files are available in the **C:\DKC3\Long Programming** folder for each problem. These may be used to test your programs.

- Each program must read from its respective input file in the **C:\DKC3\Long Programming** folder and generate an output file in that same folder. The names of these files must match what is specified for each problem in this document. These output files will be retrieved electronically and scored by the judges at the end of the session.

- Please store your source code in the **C:\DKC3** folder. Do **NOT** store any code in the **C:\DKC3\Long Programming** subfolder. Anything stored there will be overwritten.

- At the end of the session, judges will overwrite the example input files with the official input files for each problem. Each file will contain 10 test cases. One member of each team will be asked by a competition organizer to run all completed programs. Programs may be recompiled at this point prior to being run, but no changes can be made to source code.

- Each program must complete execution within **two** minutes.

- After running your programs, be sure to close out of all IDEs.

DKC^3 2020 – Long Programming Problems

1. Mars Movers (75 points)

The newest Mars rover, Digiosity, has made an amazing discovery. With its ground-penetrating sonar it has discovered a series of artifacts buried under the Martian soil. They appear to be metallic and are all in geometric shapes, and so are evidence of the planet having been inhabited, or at least visited, by some alien race sometime in the past. The shapes are squares, circles, rectangles and equilateral triangles. NASA wants to get as many of these artifacts as possible back to Earth for study.

The problem is that there isn't much storage space inside Digiosity for the return trip to Earth. Its storage compartment is only able to handle 1 layer of the artifacts and the width and height are only so large. Also, researchers are interested in studying as many different shaped objects as possible. This means if 9 small squares would fit in the storage space, but you could instead fit 5 small square shaped objects and 1 large circular object, NASA would prefer the second option. You must fit as many objects as possible while including as many different shapes as possible.

In Example 1 below, you would take the small circle and then fill in the rest of the space with squares. In Example 2 below, Triangle1 and Square1 will not fit. That leaves you with Circle1, Circle2 and Triangle2. You can either fit only Circle2 or you can fit both Circle1 and Triangle2. You'll want to do the latter.

Input
There will be ten test cases. The first line of each test case will be two whole numbers, $1 \le a \le 1000$ and $1 \le b \le 1000$, separated by a space. This represents the two-dimensional size of the storage space. Each of the next lines of the test case will consist of the name of a shape and its size separated by single spaces where $1 \le \text{size} \le 1000$ and is always an integer.

For example:

• Square1 10	*The number 10 represents the length of 1 side.*
• Square2 7	*The number 7 represents the length of 1 side.*
• Circle1 6	*The number 6 represents the diameter of the circle*.
• Triangle1 11	*The number 11 represents the length of 1 side.*
• Rectangle1 3 7	*The numbers after a rectangle represent the lengths of the 2 sides.*

There will be no more than 100 shapes per test case. Some formulas that may help:

Area of a circle = (πr^2)
Height of an equilateral triangle = (side length/2 * $\sqrt{3}$)
Area of a square = (You should already know this.)
Pythagorean Theorem = ($a^2 + b^2 = c^2$)
Area of a rectangle = (You should already know this too.)
Simple Interest Calculation = ($A = P(1 + rt)$)

Output
For each test case, list the artifacts in alphabetical order that should be placed in the storage space. Output a line containing an asterisk (*) after each test case. Clarification example: If you can only fit two squares and your square artifacts are all interchangeable and named Square1, Square2, Square 3

and Square4, then you would include Square1 and Square2 as they are alphabetically before Square3 and Square4. Every test case will have at least one artifact that will fit.

Input File: C:\DKC3\Long Programming\MarsIn.txt
Output File: C:\DKC3\Long Programming\MarsOut.txt

Examples:

Input:

```
100 100
Square1 10
Square2 10
Square3 10
Square4 10
Square5 10
Square6 10
Square7 10
Square8 10
Square9 10
Square10 10
Circle1 10
Circle2 30
40 40
Triangle1 99
Circle1 10
Triangle2 10
Square1 223
Circle2 40
```

Output:

```
Circle1
Square1
Square2
Square3
Square4
Square5
Square6
Square7
Square8
Square9
*
Circle1
Triangle2
*
```

DKC^3 2020 – Long Programming Problems

2. Word Search (75 points)

You will be given a grid of letters. Hidden within the letters are 10 words. They may be in any direction, but they will always be in a straight line, much like a pen and paper word search. Write a program that finds each of the words and outputs the location of the starting letter and a direction. The direction should be one of the following:

- Up
- Down
- Forward
- Backward
- Diagonal Up Forward
- Diagonal Up Backward
- Diagonal Down Forward
- Diagonal Down Backward

Input
Input will consist of one 25x25 grid of letters, followed by 10 words, one per line. The 10 words represent the ten test cases for the problem.

Output
For each test case, output the location of the first letter in the form (Row,Column) followed by a space and the direction needed to find the rest of the word. (0,0) is the first row, first column starting in the upper left.

Input File: C:\DKC3\Long Programming\WordSearchIn.txt
Output File: C:\DKC3\Long Programming\WordSearchOut.txt

Input/Output examples on the next page.

Examples:

Input:

```
ABATHATXYZABCDEFGHIJKXYZW

HJKSDOGSTDFGJOKLMNOPQURST

TDSFGABCDEZXYDEFHIJQURSTV

WARGHFGHIJVKLMNOBKDORJSMG

RTCMNKLMNOJOVIDKRMOMZDUSQ

ASKDJFLKJSDFLKJASLDFJLSKD

BSDFJSKJFLKSDJFLSJDFLKJSL

CNCVIOQIENVIOEWOIENVNASEF

DVMNXQWEIRUWOIEURWOIEUOIR

ELJFLKAJSLKAJSDLFKJASDFKJ

FALSKJFIELASEIJLJEKJFKASE

GZZZZZZZZZZZZZZZZZZZZZZZZ

AXXXXXXXXXXXXXXXXXXXXXXXX

DYYYYYYYYYYYYYYYYYYYYYYYY

EZZZZZZZZZZZZZZZZZZZZZZZZ

SSJDLFKJSFRODOFJDFJSLKJFL

EFLKFBILBOSDLFKJSLKFSKDJF

DGANDALFDKFJSLFJLSJFLSKDJ

BATH

CAT
```

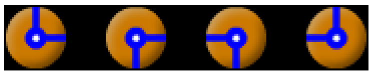
Output:

```
(0,1) Forward
(4,2) Diagonal Up Backward
```

DKC^3 2020 – Long Programming Problems

3. Spinning Wheels (75 points)

Spinning Wheels is a game for one player. It is played on a square grid consisting of **N** rows and **N** columns. Each cell contains a *wheel*, which is colored into one of five colors -- **R**ed, **O**range, **Y**ellow, **B**lue or **V**iolet (for convenience, we'll utilize the first letter of each color). Each wheel also has two *needles*, and the needles are placed in one of the following positions numbered 1 - 4:



Basically, position **1** means that the needles are directed **up** and **right**, position **2** means that they're directed **down** and **right**, position **3** means that they're **down** and **left**, and position **4** is **up** and **left**.

The goal of the game is to make all the wheels colored into the same color through a sequence of ***moves***. A move consists of choosing a wheel and rotating it 90 degrees clockwise (for example, if the wheel was in position 1, it would change to position 2). After the rotation of some wheel, the state of other wheels (as well as the state of the wheel rotated first) may change.

More formally. Let's denote the color of the wheel chosen on this move by **C**. We'll say that two wheels are ***matching*** if they are situated in side-by-side neighboring cells and either needle of both wheels is directed to the other wheel. The process of changing the grid is separated into ***iterations***. We will maintain set **S** consisting of the wheels rotated at the last iteration. Before the first iteration, set **S** contains only the starting wheel. The following algorithm is executed then repeatedly:

- If set **S** is empty, stop the process.
- Rotate each wheel in set **S** 90 degrees clockwise.
- Assign color **C** to each wheel in set **S**.
- Form set **Q** consisting of wheels which have at least one matching wheel in set **S**.
- Replace set **S** with set **Q** and clear set **Q**.

It's possible to prove that this process will definitely stop. Note that a wheel may get rotated more than once during the move.

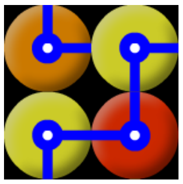
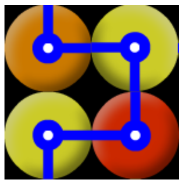
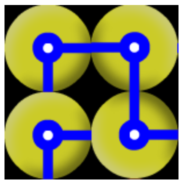
Normally, the goal is to complete the game in as few moves as possible to fill the board with a single color. For the purpose of this problem, the goal is simply to read in a set of moves and track the iterations and output a specific set of values.

Input/Output information on the following pages.

Input
Each file will consist of 10 test cases, and each test case starts with a number **N** where **2 <= N <= 1000**. The following **N** lines contain **N** integers (between 1 and 4, inclusive) each and denote the positions of the wheels in the corresponding cells. The next following **N** lines give the starting colors (R, O, Y, B, V) in the same format. These positional integers and color characters are all separated spaces. The final line of each test case will consist of an array of moves formatted as an array of Row/Column coordinates with an upper left origin. *Due to the potential sizes of these test cases, it is important to keep performance in mind!*

```
2
1 2
2 4
O Y
Y R
[(0,1)]
```

Start **Move** **Iteration**

Start: Initial positioning and color.
Move: Wheel at position (0, 1) is rotated 90 degrees clockwise and now aligns with (0, 0) and (1, 1).
Iterations: The newly aligned wheels at (0, 0) and (1, 1) each rotate 90 degrees clockwise and change color to match. **This is 1 iteration.** If the wheel at (1, 0) had aligned with the wheel at (0, 0) in the 1st iteration, then the wheel at (0, 0) would rotate in an additional iteration. Any sequence of rotations is an iteration, including the one that starts the move.

Output
For each test case output, output the final position and colors of the wheels in the same format as they were input. **N** lines that contain **N** integers which denote position, and **N** lines of **N** characters (R, O, Y, B, V) for color. The line following the color matrix will be the sum of all ***iterations*** performed throughout each ***move***. *Completing the moves or winning the game with all matching colors will end the process.*

```
2 3
2 1
Y Y
Y Y
2
```

Input File:	C:\DKC3\Long Programming\SpinningWheelsIn.txt
Output File:	C:\DKC3\Long Programming\SpinningWheelsOut.txt

Examples:

Input:

```
2
1 2
2 4
O Y
Y R
[(0,1)]
4
3 2 2 4
4 4 1 4
1 3 1 2
1 2 3 4
O Y Y O
V B O O
O B V R
R B B Y
[(3,3),(0,0)]
```

Output:

```
2 3
2 1
Y Y
Y Y
2
4 2 2 4
4 4 1 4
1 3 3 4
1 2 3 2
O Y Y O
V B O O
O B Y Y
R B B Y
6
```

DKC^3 2020 – Long Programming Problems

4. Spare Shopping (75 points)

Every week Jean shops for items called spares, which come in one of ten colors: red, yellow, green, blue, orange, purple, fuchsia, jade, white, and magenta. However, shopping for spares is not easy because the store only offers packages of multiple spares. This is complicated further because the packages offered, and their prices, change from week to week. Jean is looking for a way to determine which packages to buy to get the spares she needs each week while spending the least amount of money possible.

You will be given a list of the packages offered by the spare store, as well as the list of spares Jean needs that week. You must determine which packages (and how many) Jean must buy to get what she needs at the cheapest price. She is okay with getting more spares than she needs as long as she gets everything on her list. It will always be possible for Jean to get everything on her list.

Input
There are ten test cases. Each test case will consist of two lines. The first line will detail some number of packages offered by the spare store. The details of each package will come in three parts – the package name, the package contents and the price, each separated by a space. For example:

```
P1 2R5P1W $10.50
```

This example package is called P1 and includes 2 red, 5 purple and 1 white spares for $10.50. Each package delineation will be separated by a comma and space. The second line of the input will be a list of which spares Jean needs, listed similarly to the package details (for example, 4Y4F8M means that Jean needs 4 yellow, 4 fuchsia, and 8 magenta spares).

Output
For the output, you must print out a comma-separated list of which packages Jean should buy, indicating the number of each parenthetically. Print out packages Jean should buy in the same order they are listed in the input.

Input File: C:\DKC3\Long Programming\SpareShoppingIn.txt
Output File: C:\DKC3\Long Programming\SpareShoppingOut.txt

Examples:

Input:
```
P1 2R2Y1J $4.50, P2 5W5J $12.00, P3 1R1Y1G1B1O1P1F1J1W1M $2.50
6R6Y5J
P1 1Y1G1O $3.00, P2 1Y1M $2.00, P3 2M $5.00, P4 3O3P $7.20, BONUSPACKAGE 2G2W2M $2.50
10Y8G6O4M
```

Output:
```
P1(1), P2(4)
P1(6), P2(4), BONUSPACKAGE(1)
```