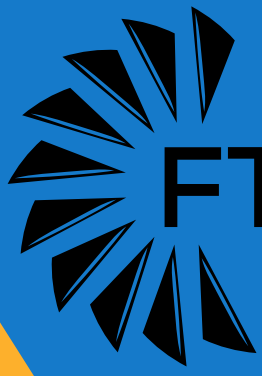
THE CRC ROBOTICS SENIOR PROGRAMMING PROBLEMS

PRELIMINARY 4

MODUEL 2026

presented by



FTAI AVIATION

A program of

AEST EAST

Version 1.0

A FEW NOTES

- The complete rules are in section 4 of the rulebook.
- You have until **Friday, February 13th, 11:59 pm** to submit your code.
- Feel free to use the programming forum on the CRC discord to ask questions and discuss the problem with other teams. It is there for that purpose!
- **We are giving you quick and easy-to-use template files for your code and the tests. You are required to use them.**

USING THE TEMPLATE FILE

- The template tests call the function to test, take the output and allow you to quickly check if your code works as intended. **All your code, except additional functions you create, should be written in the function of the part of the problem you are solving.**
- Points given in the document are indications of how difficult the section is and how many points you will get if you complete it. This preliminary problem is going to be 2% of the main challenge towards the global score of the programming competition and for more points related information consult the rulebook.

STRUCTURE

Every problem contains a small introduction like this about the basics of the problem and what is required to solve it.

Input and output specification:

Contains the inputs and their format, and which outputs the code is required to produce and in what format they shall be.

Sample input and output:

Contains a sample input, sometimes containing sub examples in the sample input, and what your program should return as an output.

Explanation of the first output:

Explains briefly the logic that was used to reach the first output, given the first input.

Patterns and repetitions!

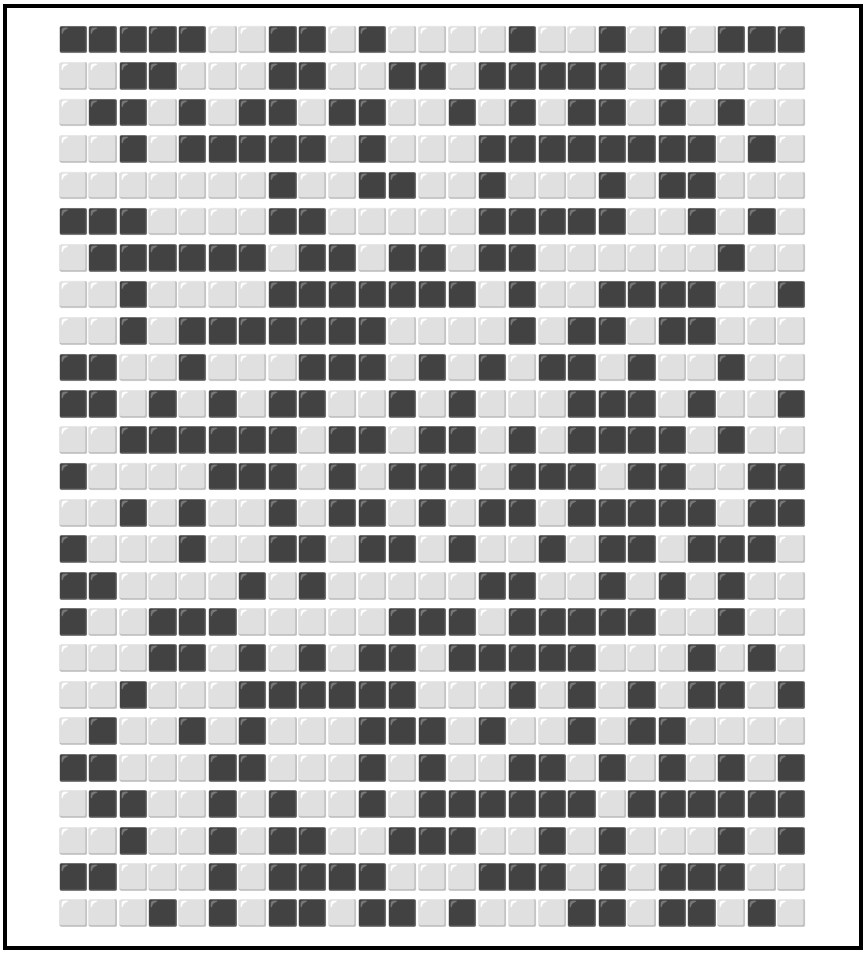
At the very ripe age of… 24. I, Frank, start to repeat myself often. Some see a problem in it, but I find some patterns and symmetry in the midst of it all. The more we look for them, the more patterns and repetition we can find in our everyday life, such as in words, numbers, tiles, mountains or even in the waves.

Part 1: Bathroom Tiles (10 points)

You've just finished renovating your bathroom, and you asked the tile installer to place the tiles in a random pattern. However, you can't help but notice the same motifs repeating themselves…

Using the layout of your bathroom tiles, you will need to count how many times a given pattern appears.

Here is the floor plan:



Input and output specification:

You will receive a *list* of *lists* (a 2D array) that represents the pattern you need to find in the bathroom grid. This pattern will be included directly in your Python file. Be careful! The 0s in the pattern indicate that either a black ***OR*** white tile can appear in that position.

You must return an *int* representing how many times the given pattern appears. It's important to note that the patterns must not overlap when you count them. It's up to you to find the optimal arrangement of the pattern in order to fit the highest possible number of occurrences.

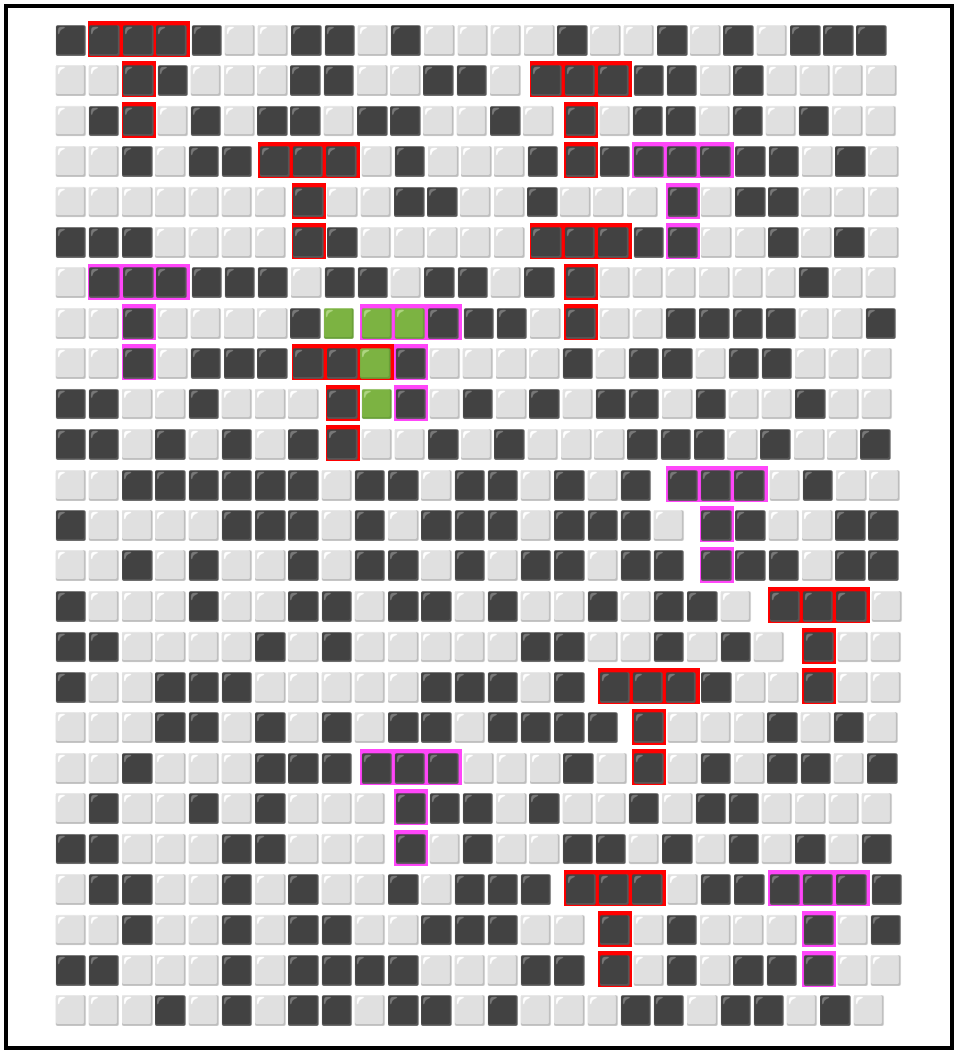
Sample input:

```
[ [1, 1, 1],
  [0, 1, 0],
  [0, 1, 0] ]

[ [0, 0, 1],
  [0, 1, 1],
  [1, 1, 0] ]

[ [1, 1],
  [1, 1] ]

[ [1, 0],
  [1, 0],
  [1, 1],
  [0, 1] ]

[ [1, 1],
  [0, 1],
  [1, 1] ]
```

Sample output

```
14
14
25
13
12
```

Explanation of the first output:



Even though we can find the pattern in other places, like the green *“T”*, only 14 of them can be placed at the same time on the floor plan.

Part 2: Palindromes (X points)

Palindromes are words, numbers or sequences of symbols that are read the same forward and backward. For example, the word *“Laval”* or the date *"12/02/2021"* are palindromes.

But for the sake of this problem, we’re gonna focus on number palindromes. You’ll have to resolve an equation you’re given and once it’s solved, you have to check if the number obtained is a palindrome. Since we are not focusing on the equation solving part too much, you don’t have to take into consideration the priority of operations, from left to right will do just fine for this problem.

Input and output specification:

You will receive the math equation as a *string* of numbers and basic arithmetic operations (+, -, x, ÷).

You have to output *Yes* if the result of the equation is a palindrome and *No* if it's not a palindrome.

Sample input:

36363×11

121×343+55255

572×471+550

254168116÷4-62999784

Sample output:

```
Yes

No

Yes

Yes
```

Explanation of the first output:

The first equation gives 399993 and backwards it still is 399993, so it is a palindrome.

Part 3: Seamless Mosaic (X points)

You are designing a mosaic floor using square tiles. Each tile is composed of 1x1 small tiles that are either black (@) or white (#) and is represented as a square grid, for example:

```
@#
#@
```

means that this is a 2x2 tile with black tiles at the top-left and bottom-right corners, and white tiles at the top-right and bottom-left corners. Similarly,

```
@@#
@##
###
```

represents a 3x3 tile, with black tiles in the top-left corner.

You are now tasked to find the most cost-effective way to create a large mosaic floor of size M x N using smaller tiles. **Remember, you are only allowed to produce *square* tiles of the *same* size.** You will also be able to rotate the design by 90, 180, or 270 degrees without needing to count the rotated pattern as another design.

You will be given two numbers, M and N, where M is the height of the mosaic floor you need to build and N is the width of the mosaic floor. You will also be given M N-character long strings, representing the desired pattern of the mosaic floor, where each character is either @ (black) or # (white).

The cost of the floor is calculated in the following way:

$$(50 + floor(\frac{360}{s}))i + 80t$$

To interpret the formula, you can consider:

- The first part, $(50 + floor(\frac{360}{s}))i$, represents the cost of designing a **type** of tile. With a base cost of 50, the larger is the tile size s, the cheaper it is to design each tile **type**. i is the number of **types** designed.
- The second part, $80t$, represents the cost of producing the tiles **themselves**. Each tile produced costs 80, and t is the total number of tiles used.

Also, note that the floor function rounds the number inside down to the nearest integer, e.g.:

$$floor(3.7) = 3$$

Your task is now to find the least amount of money needed to build the given mosaic floor and print it as the output.

Input and output specification:

- You will *receive* two integers M and N, denoting the height and width of the mosaic floor you need to build.
- You will then *receive* a list of strings with size M, each string having length N, representing the desired pattern of the mosaic floor. Only @ and # will appear in the string.

- You will then **output** a single integer, representing the amount of money needed to build the mosaic floor in the most cost-effective way.

Sample input:

```
M=3 N=3
@@#
@##
###


M=2 N=4
@@##
##@@

M=4 N=8
@@@@@#@#
@@@@#@#@
@@@@@#@#
@@@@#@#@
```

Sample output:

```
250

390

440
```

Explanation of the first output:

You have the choice of making nine 1x1 tiles, or one 3x3 tiles.

If you are making 1x1 tiles, there are 2 designs (solo black and solo white) and 9 tiles required, giving the cost:

$$(50 + 360) \times 2 + 80 \times 9 = 1540$$

In contrast, if you are making a 3x3 tile, there is only 1 design and 1 tile required, therefore the cost is:

$$(50 + 360/3) + 80 * 1 = 250$$

Therefore the most cost-effective way of building the floor is by designing a 3x3 tile and using only 1 of those, costing at the end 250.

Explanation of the second output:

You have the choice of making 1x1 tiles or 2x2 tiles.

If you are making 1x1 tiles, there are 2 designs (solo black and solo white) and 8 tiles required, giving the cost:

$$(50 + 360) \times 2 + 80 \times 8 = 1460$$

In contrast, if you are making 2x2 tiles, there is only 1 design and 2 tiles required, therefore the cost is:

$$(50 + 360/2) + 80 \times 2 = 390$$

Therefore the most cost-effective way of building the floor is by designing a 2x2 tile and use only 1 of those, costing at the end 390.

Note that there is only one pattern:

```
@@
##
```

since the right half is just this same pattern rotated 180 degrees:

```
##
@@
```

Part 4: Convex or concave? (X points)

By using a list of 2D coordinates that will be given to you, you will have to determine if it is possible to create a convex polygon by drawing lines between all the given points.

ps.: It is possible to transform the cartesian coordinates into polar coordinates. It will be up to you to decide whether or not that can be helpful or not. You can also use mathematical graphing software like GeoGebra or Desmos to help yourself visualize the lists or points to think about how you would attack this problem.

Spécification d'entrée et de sortie:

As input, you will receive a list of cartesian 2D coordinates (x, y).

As output, you will have to give a *bool* that represents whether or not it is possible to draw lines between all points to make a convex polygon.

Exemple d'entrée:

[(4, 0), (0, 4), (-4, 0), (0, -4)]
[(5, 1), (6, 4), (2, 8), (-1, 6), (-6, 8), (-8, 3)]
[(-3, 5), (0, 0), (10, 0), (-7, 5), (3, 5), (-10, 0), (0, -10), (7, 5)]
[(-6, -15), (3, 3), (-2, 3), (4, 18), (-8, 0), (-4, 0)]

Exemple de sortie:

```
True
False
False
False
```

Explication de la première sortie:

Use hand made drawing and other graphic tools to help yourself. Many solutions are possible, use your creativity and develop your problem solving skills.