N3TWORK Chile SpA
Nueva Tajamar 481, Of 1102
Torre Sur, Las Condes

Software Engineering Test

Thank you for taking the time to complete our programming test. You have 2 hours to send your first results over email to us. The time stamp of your email will be used to time the test. We will not consider your test if you don't send your first update within 2 hours. You can take up to 3 more hours to finish the test for a total of 5 hours.

It is ok to send a partial answer if you can't finish the test. We will evaluate your answers and may discuss further in your next interview. If you have questions during the test, you can send an email to sebastian@n3twork.com.

Please write your answers in the language that you were told to. You can write your own helper functions and use the standard library, but not use any other external libraries.

Please make sure your code compiles, run and generates the requested output when specified. When ready, pack your source code in a zip file and email it back.

Copy&Paste from external sources is not allowed, please write your own functions.

Spiral

Write a function named **printMatrix**, that takes an array of integers representing a 2d square matrix and the number of elements on each side as parameters. This function should start at the upper left hand corner of the matrix and print each element in a counterclockwise spiral.

Example: The 3x3 matrix:

```
| 1 8 7 |
| 2 9 6 |
| 3 4 5 |
```

Is represented by the array **m**, and the size **n**:

```
m = [1, 8, 7, 2, 9, 6, 3, 4, 5]
n = 3
```

The result of executing the function **printMatrix** with **m** and **n** as parameters is:

```
1 2 3 4 5 6 7 8 9
```

Valid values for the elements of **m** and **n** are:

```
0 <= a ∈ m <= 2^31-1
1 <= n <= 2^15
```

Primes

- Write a function named **printNPrimes**, that takes an **int** representing the amount of prime numbers to print.

  For example: **printNPrimers** with the parameter **5** will print:
  `2 3 5 7 11`

- Propose how would you refactor your implementation to work when receiving a **long** parameter with value `MAX_LONG: 2^63 - 1`. You don't have to implement it.

Circles

- Write a function named **findCircleCollisions** that takes an array of circles as parameter and computes a new array with the circles that are colliding. A collision is defined by a positive intersection area between two circles.

- Write a new function named **findCollisions** that takes an array of shapes, including circles. You should leave the shape collision math to future implementors.