Tercer Reto.

Más información sobre Expresiones Regulares.

La empresa para la que trabajas empezará a desarrollar un parser de expresiones regulares hace tiempo, pero el programador principal y el jefe del proyecto, tuvieron un importante enfrentamiento por la manera que el programador tenía de hacerlo. Los dos fueron incapaces de resolver sus diferencias y el programador fue despedido. Enfadado de que le despidieran injustamente, empezó a insultar al jefe y borró prácticamente la mayoría del programa de manera irreversible, lo que complica la situación mucho más.

Tu jefe, en cambio te pide que hagas el parser desde el principio. Le disgusta tanto la idea de pensar en el programador que despidió, que te prohibe hacerlo empleando cualquier estructura de datos clásica simplemente por orgullo personal.

El programa deberá recibir por entrada de teclado una cadena de caracteres y una expresión regular y comprobar si la cadena *completa* cuadra con la expresión. El parser deberá tener en cuenta, **en este orden**:

```
Letras mayúsculas y minúsculas: "a-z", "A-Z"
Números:                        "0" a "9"
Símbolos de delimitación:       "[]","()", "-"
Repetición:                     "+", "*"
Indeterminación:                "?"
```

NOTA INFORMATIVA: UNA VEZ FUNCIONE UNO VENIR A ENTREGARLO. **NO** ESPERAR A QUE FUNCIONEN TODOS

Ejemplos:

```
Expresión: ab
    Input: ab
   Output: "Correcto"
---------------------------
Expresión: [a-z][0-9]
    Input: sh
   Output: "Incorrecto"
```

-**Restricciones**: Queda prohibido emplear **cualquier estructura de datos clásica para almacenar los símbolos del input.** Se considerará “estructura de datos clásica” cualquiera de la siguiente lista: **array, linked-list, stack, queue, tree, graph, table, set; o variantes que imiten explícitamente el funcionamiento de estas**. Independientemente del método que se emplee para analizar el input, **el backtracking estÃ¡ prohibido**.

Lenguajes aceptados: C/C++, Python & Java.

Duración: 1h 30m.
