Índice

Ejercicios Unidad 11

Descargar estos ejercicios

> **Antes de empezar**

Para realizar estos ejercicios, deberás descargar los recursos del enlace de proyecto_expresiones_regulares. Como puedes ver, la solución está compuesta de varios proyectos. Cada uno de ellos corresponde con un ejercicio, deberás implementar todo el código, tanto de la Main como de los métodos que se piden en cada ejercicio. Cada proyecto contiene el test correspondiente, que deberás pasar para comprobar que has hecho el ejercicio correctamente. Estos ejercicios están pensados para trabajar de forma avanzada el manejo de EERR en C#. Se recomienda revisar los apuntes de la unidad antes de resolverlos.

Ejercicio 1. Buscador de etiquetas

Implementa un método que permita extraer contenido entre etiquetas HTML (no contemples la posibilidad de etiquetas anidadas). Usa expresiones regulares para capturar todos los bloques `<etiqueta>...</etiqueta>` válidos.

Salida por consola:

```
ExtraeEtiquetas("<p>Hola mundo</p><p>¿Qué tal estás?</p>")
// Devuelve: ["Hola mundo", "¿Qué tal estás?"]
```

Requisitos:

- Utiliza expresiones regulares con grupos de captura.
- Debe funcionar con etiquetas sin anidar y diferentes tipos de etiquetas.
- No utilices métodos manuales para buscar los delimitadores.

Ejercicio 2. Reescritor Tiko extendido (versión con expresiones regulares)

En esta segunda parte vas a mejorar el traductor al lenguaje Tiko permitiendo el uso de **expresiones regulares (Regex)** para hacer el procesamiento más potente, limpio y flexible.

A partir de una frase introducida por el usuario, realiza las siguientes transformaciones, ahora utilizando `Regex` en los puntos señalados:

1. ✅ Traducción de dígitos a texto
2. ✅ Eliminación de duplicados de letras consecutivas
3. ✅ Sustitución de onomatopeyas por emojis
4. ✅ Transformaciones adicionales opcionales

Salida por consola:

```
Introduce una frase: jajajaja tengo 2 perros y holaaa x q no vienes?
Tiko extendido: 😂 tengo dos perros y hola por qué no vienes?
```

Requisitos:

- Usa `Regex.Replace()`.
- Emplea correctamente los grupos de captura y los cuantificadores (`+` , `*` , `{n,m}`).
- No uses métodos manuales (`foreach` , `if` , `for`) para las transformaciones que pueden resolverse con expresiones regulares.
- Utiliza `StringBuilder` solo si lo necesitas para la aleatorización final de mayúsculas.

Ejercicio 3. Evaluador de expresiones aritméticas simples

Crea un evaluador de expresiones tipo “2+3*4-1”. Usa expresiones regulares para separar los números y operadores, y luego calcula el resultado respetando la prioridad de operadores.

Salida por consola:

```
Introduce una expresión: 2+3*4-1
Resultado: 13
```

Requisitos:

- Utiliza expresiones regulares para separar operandos y operadores.
- Debe respetar la prioridad de operadores: multiplicación y división antes que suma y resta.
- No utilices `eval` ni métodos de ejecución directa de código.

Ejercicio 4. Analizador de logs

Dado un conjunto de líneas de log con marcas de tiempo tipo `[2025-07-28 12:34:56]`, extrae:

- Todos los errores (`ERROR:`)
- Todos los eventos únicos por tipo (INFO, WARN, etc.)
- El primer y último mensaje

Salida por consola:

```
AnalizarLogs([
  "[2025-07-28 12:34:56] INFO: Inicio",
  "[2025-07-28 12:35:00] ERROR: Fallo de conexión",
  "[2025-07-28 12:36:00] WARN: Memoria baja",
  "[2025-07-28 12:37:00] INFO: Fin"
])
Errores encontrados:
Fallo de conexión
Tipos de eventos únicos:
INFO, ERROR, WARN
Primer mensaje: Inicio
Último mensaje: Fin
```

Requisitos:

- Utiliza expresiones regulares para extraer los tipos de eventos y mensajes.
- Debe identificar el primer y último mensaje del log.
- Presenta los resultados en listas separadas.

Ejercicio 5. Creador de etiquetas Markdown

Crea un método que convierta un texto plano con delimitadores personalizados en formato Markdown. Ejemplo:

- `**negrita**` → `<strong>`
- `_cursiva_` → `<em>`
- `[enlace](url)` → `<a href="url">enlace</a>`

Salida por consola:

```
Introduce un texto: Esto es **importante** y _cursivo_ y [Google](https://google.com)
Convertido: Esto es            y cursivo y Google
```

Requisitos:

- Utiliza expresiones regulares para detectar los delimitadores.
- Convierte los formatos Markdown a HTML.
- Debe funcionar con varios formatos en la misma cadena.