Ejercicio 1. A partir de la lista genérica implementada en prácticas y los conceptos explicados en clase, crear una clase `Cola` genérica cuyo constructor reciba un `IEnumerable` de predicados que no podrá ser modificado. Deben implementarse los siguientes métodos y/o propiedades:

- `EstaVacía`: Devolverá si quedan o no quedan elementos en la cola.
- `Encolar`: Insertará al final de la cola el elemento pasado como parámetro solamente si cumple con todos los predicados de la cola. Devolverá si la operación ha tenido éxito o no.
- `Desencolar`: Eliminará el elemento más antiguo de la cola y lo devolverá. El tipo de retorno de este método debe ser `void`.

Crear un proyecto de test que pruebe la funcionalidad del apartado anterior.

Ejercicio 2. Implementar un método extensor de `IEnumerable` llamado `EncodeRLE` capaz de comprimir un `IEnumerable` usando el algoritmo Run-Lenght Encoding (RLE). Este algoritmo de compresión reduce el tamaño de una secuencia con repeticiones consecutivas del mismo valor reemplazándolas por un único valor y su recuento. Por ejemplo, supongamos una secuencia de pixeles negros y blancos que queremos comprimir.

```
"BBBBBBBBBBBBNBBBBBBBBBBBBNNNBBBBBBBBBBBBBBBBBBBBBBBNBBBBBBBBBBBBB"
```

Si aplicamos Run-Lenght Encoding, esta secuencia pasaría a ser (en pseudo-código):

```
[(12, B), (1, N), (12, B), (3, N), (24, B), (1, N), (14, B)]
```

Implementar el método inverso, llamado `DecodeRLE`, que dado un `IEnumerable` de objetos `Tupla`, devuelve el `IEnumerable` original.

Ejercicio 3. A partir del modelo utilizado en la sesión 8 de prácticas y empleando *LINQ*:

- Para cada edificio, obtener la edad máxima, la edad mínima y el numero de empleados que tienen despacho en ese edificio.
- Mostrar las llamadas realizadas desde el edificio "Faculty of Science" al edificio "Polytechnical". Debe mostrarse por pantalla el número de origen, el número de destino y la duración de la llamada.

Ejercicio 4. Sin usar *LINQ*, implementar un método perezoso llamado `ZipLongest` que recibe

- `seqLeft`: La primera secuencia (`IEnumerable`).
- `seqRight`: La segunda secuencia (`IEnumerable`).
- `defLeft`: Por defecto tiene como valor el "valor por defecto" de su tipo.
- `defRight`: Por defecto tiene como valor el "valor por defecto" de su tipo.

Este método combina los elementos de `seqLeft` y `seqRight` y los devuelve en un tercer `IEnumerable`. Es decir, combina el primer elemento de cada secuencia, luego el segundo elemento y así sucesivamente. Si en algún momento, alguno de las dos secuencias se queda sin elementos, se ha de usar el `defLeft` o `defRight`. Cuando ambas secuencias se quedan sin elementos, el método termina.

Ejemplos (en pseudo-código) con secuencias de enteros y cadenas:

- `ZipLongest([ 1, 2 ], [ "a", "b", "c"])`
  devuelve `[(1, "a"), (2, "b"), (0, "c")]`
- `ZipLongest([ 1, 2, 3 ], [ "a", "b"])`
  devuelve `[(1, "a"), (2, "b"), (3, null)]`
- `ZipLongest([ 1, 2 ], [ "a", "b", "c"], defLeft: 1000)`
  devuelve `[(1, "a"), (2, "b"), (1000, "c")]`
- `ZipLongest([ 1, 2, 3 ], [ "a", "b"], defRight: "DEF")`
  devuelve `[(1, "a"), (2, "b"), (3, "DEF")]`