```
        Console.WriteLine( palabra );
      }

    }

    // Constructor
    public PilaString() {
      posicionPila = 0;
      datosPila = new string[MAXPILA];
    }

    // Añadir a la pila: Apilar
    public void Apilar(string nuevoDato) {
      if (posicionPila == MAXPILA)
        Console.WriteLine("Pila llena!");
      else {
        datosPila[posicionPila] = nuevoDato;
        posicionPila ++;
      }
    }

    // Extraer de la pila: Desapilar
    public string Desapilar()  {
      if (posicionPila < 0)
        Console.WriteLine("Pila vacia!");
      else {
        posicionPila --;
        return datosPila[posicionPila];
      }
      return null;
    }

}  // Fin de la clase
```

**Ejercicios propuestos:**

- **(11.7.1)** Usando esta misma estructura de programa, crear una clase "Cola", que permita introducir datos (números enteros) y obtenerlos en modo FIFO (el primer dato que se introduzca debe ser el primero que se obtenga). Debe tener un método "Encolar" y otro "Desencolar".
- **(11.7.2)** Crear una clase "ListaOrdenada", que almacene un único dato (no un par clave-valor como los SortedList). Debe contener un método "Insertar", que añadirá un nuevo dato en orden en el array, y un "Extraer(n)", que obtenga un elemento de la lista (el número "n"). Deberá almacenar "strings".
- **(11.7.3)** Crea una pila de "doubles", usando internamente un ArrayList en vez de un array.
- **(11.7.4)** Crea una cola que almacene un bloque de datos (struct, con los campos que tú elijas) usando un ArrayList.
- **(11.7.5)** Crea una lista ordenada (de "strings") usando un ArrayList.

## *11.8. Introducción a los "generics"*

Una ventaja, pero también a la vez un inconveniente, de las estructuras dinámicas que hemos visto, es que permiten guardar datos de cualquier tipo, incluso datos de distinto tipo en una

misma estructura: un ArrayList que contenga primero un string, luego un número entero, luego uno de coma flotante, después un struct… Esto obliga a que hagamos una "conversión de tipos" con cada dato que obtenemos (excepto con los "strings").

En ocasiones puede ser interesante algo un poco más rígido, que con las ventajas de un ArrayList (crecimiento dinámico, múltiples métodos disponibles) esté adaptado a un tipo de datos, y no necesite una conversión de tipos cada vez que extraigamos un dato.

Por ello, en la versión 2 de la "plataforma .Net" se introdujeron los "generics", que definen estructuras de datos genéricas, que nosotros podemos particularizar en cada uso. Por ejemplo, una lista de strings se definiría con:

```
List<string> miLista = new List<string>();
```

Y necesitaríamos incluir un nuevo "using" al principio del programa:

```
using System.Collections.Generic;
```

Con sólo estos dos cambios, el ejemplo de uso de ArrayList que vimos en el apartado 11.4.1 funcionaría perfectamente.

De esta misma forma, podríamos crear una lista de structs, o de objetos, o de cualquier otro dato.

No sólo tenemos listas. Por ejemplo, también existe un tipo "Dictionary", que equivale a una tabla Hash, pero en la que las claves y los valores no tienen por qué ser strings, sino el tipo de datos que nosotros decidamos. Por ejemplo, podemos usar una cadena como clave, pero un número entero como valor obtenido:

```
Dictionary<string, int> dict = new Dictionary<string, int>();
```

**Ejercicios propuestos:**

- **(11.8.1)** Crea una nueva versión de la "bases de datos de ficheros" (ejemplo 46), pero usando List en vez de un array convencional.
- **(11.8.2)** Crea un programa que lea todo el contenido de un fichero, lo guarde en una lista de strings y luego lo muestre en orden inverso (de la última línea a la primera).

## *11.9. Los punteros en C#.*

### 11.9.1. ¿Qué es un puntero?

La palabra "puntero" se usa para referirse a una **dirección de memoria**. Lo que tiene de especial es que normalmente un puntero tendrá un tipo de datos asociado: por ejemplo, un "puntero a entero" será una dirección de memoria en la que habrá almacenado (o podremos almacenar) un número entero.