Programación

C#

TECNICATURA SUPERIOR EN PROGRAMACIÓN
LABORATORIO DE COMPUTACIÓN II

UTN
Facultad Regional
San Francisco

Clase StringBuilder

- Representa una cadena de caracteres modificable.
  - Puede cambiarse una vez se ha creado mediante adición, sustracción, sustitución o inserción de caracteres.

```
StringBuilder MiStringBuilder = new StringBuilder("Hello World!”);
```

- La mayoría de los métodos que modifican una instancia de esta clase, devuelven una referencia a esa misma instancia.
- Se puede utilizar cuando se desee modificar una cadena sin crear un nuevo objeto.

FUNCIONALIDAD

- Proporciona métodos para:
  - Modificar las cadenas: agregar, insertar, quitar, reemplazar.
  - Consultar y definir la longitud y capacidad de las cadenas.

Clase StringBuilder

funcionalidad

Capacidad y Longitud

- **Capacidad**: número máximo de caracteres que puede contener la cadena.
- **Longitud**: Obtiene o establece la longitud del objeto actual.
  - Por ejemplo, se crea un objeto StringBuilder con la cadena "Hello", que tiene una longitud de 5, y especificar que el objeto tenga una capacidad máxima de 25.

```
StringBuilder MyStringBuilder = new StringBuilder("Hello World!", 25);
MyStringBuilder.Capacity = 30; //Aquí se aumenta la capacidad
```

- La capacidad *no puede ser menor* que el valor de la propiedad Length.

Clase StringBuilder

funcionalidad

Modificar una Cadena

- **Append**
  - Permite agregar texto al final de una cadena. El espacio se asigna automáticamente según sea necesario. Por ejemplo:

```
StringBuilder MyStringBuilder = new StringBuilder("¡Hola Mundo!");
MyStringBuilder.Append(" Qué lindo día.");
Console.WriteLine(MyStringBuilder); // "¡Hola mundo! Qué lindo día."
```

- **AppendFormat**
  - Igual que Append, pero acepta cadenas de formato que contienen cero o más especificaciones de formato. Por ejemplo:

```
int MyInt = 25;
StringBuilder MyStringBuilder = new StringBuilder("Su total es");
MyStringBuilder.AppendFormat("${0} ", MyInt);
Console.WriteLine(MyStringBuilder); // Salida: "Su total es $25"
```

Modificar una Cadena

- **Ejercicio 1**
  - Crear un programa que muestre por pantalla lo siguiente:

    Lógicamente

    Lógicamente ahora

    Lógicamente ahora lo

    Lógicamente ahora lo urgente

    Lógicamente ahora lo urgente es

    Lógicamente ahora lo urgente es esperar

    Debe utilizar un vector que contenga las palabras antes especificadas y a partir de ello generar lo que debe mostrarse.

Clase StringBuilder

funcionalidad

Modificar una Cadena

- **Insert**
  - Agrega una cadena en una posición específica en la StringBuilder actual.

```
StringBuilder MyStringBuilder = new StringBuilder("¡Hola mundo!");
MyStringBuilder.Insert(5,"hermoso ");
Console.WriteLine(MyStringBuilder); // Salida: "¡Hola hermoso mundo!"
```

- **Ejercicio 2**
  - Crear un programa que muestre por pantalla lo siguiente:

esperar
es esperar
urgente es esperar
lo urgente es esperar
ahora lo urgente es esperar
Lógicamente ahora lo urgente es esperar

Debe utilizar un vector que contenga cada una de las palabras mencionadas.

Clase StringBuilder

funcionalidad

Modificar una Cadena

- **Remove**
  - Permite quitar cierta cantidad de caracteres comenzando en la posición indicada.

```
StringBuilder MyStringBuilder = new StringBuilder("¡Hola Mundo!");
MyStringBuilder.Remove(5,7);
Console.WriteLine(MyStringBuilder);  // Salida: "¡Hola"
```

- **Replace**
  - Reemplaza todas las apariciones de una cadena o caracter en el objeto StringBuilder con otra cadena o caracter.

```
StringBuilder MyStringBuilder = new StringBuilder("!Hola Mundo!");
MyStringBuilder.Replace('!', '?');
Console.WriteLine(MyStringBuilder);  // Salida: "?Hola mundo?"
```

Modificar una Cadena

- **Ejercicio 3**
  - Modificar el primer ejercicio para que a además se imprima lo siguiente:

    Lógicamente ahora lo urgente es

    Lógicamente ahora lo urgente

    Lógicamente ahora lo

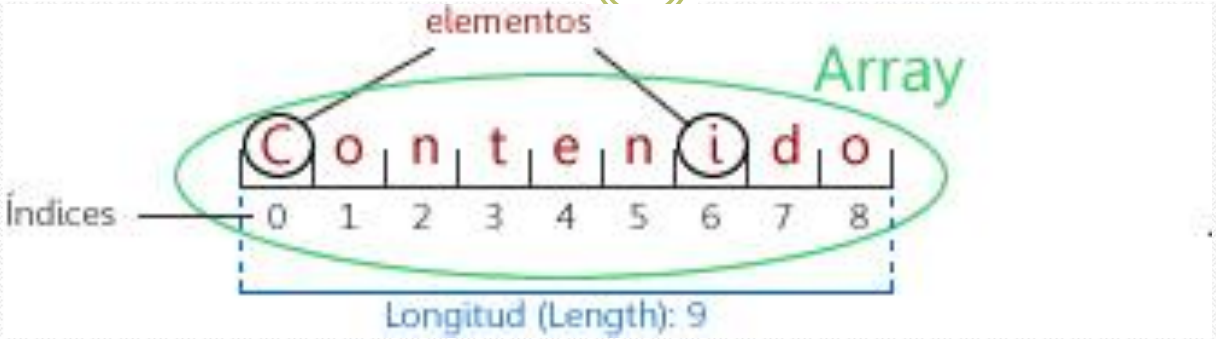
    Lógicamente ahora

    Lógicamente

Debe utilizar un vector que contenga las palabras antes especificadas.

- Ejercicio 4: Realice el ejercicio 3 utilizando los métodos Remove y luego Replace.

Clase Array



- Es un conjunto de elementos del **mismo tipo**, que comparten un **nombre común** y a los que se puede acceder a través de su posición (**índice**) que cada uno de ellos ocupa en el arreglo. Por ejemplo:

```
int[] numeros = new int[5] { 1, 2, 3, 4, 5 };
string[] diasSemana = new string[] {"lunes", "martes", "miercoles", "jueves", "viernes",
    "sabado", "domingo"}
```

- Proporciona métodos para la creación, manipulación, búsqueda y ordenación de matrices.

- Para situaciones en las que el acceso a los datos se realice de forma aleatoria e impredecible.

FUNCIONALIDAD

- Proporciona métodos para:
  - Creación,
  - manipulación,
  - búsqueda y
  - ordenación de matrices

Clase Array

funcionalidad

Crear y acceder a un Array

- **Crear un arreglo**
  - Implica reservar la cantidad de memoria necesaria para contener todos sus elementos y asignar al nombre del arreglo una referencia a ese bloque. Por ejemplo:

```
int[] m = new int[10]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
string[] diasSemana = new string[] {"lunes", "martes", "miercoles", "jueves",
    "viernes", "sabado", "domingo"}
```

- **Acceder a los elementos**
  - Se utiliza el nombre del arreglo seguido de un subíndice entre corchetes. Por ejemplo:

```
int[] numeros = new int[5] { 1, 2, 3, 4, 5 };
int suma = 0, indice;
for (indice = 0; indice < numeros.Length; indice++)
{
    suma += numeros[indice];
} // La suma total es 15.
```

Clase Array

funcionalidad

ORDENACIÓN

- **Sort**
  - Ordena los elementos de toda una matriz unidimensional. Por ejemplo:

```
string[] palabras = {"el", "verde", "lindo", "queda", "salta",
   "ejercitación", "y", "mira"};
Array.Sort(palabras);
// Salida: ejercitación – el – lindo – mira – queda – salta – verde – y
```

- **Reverse**
  - **Invierte el orden** de los elementos de una matriz Array unidimensional o de una parte de Array. Por ejemplo, al arreglo anterior se le aplica:

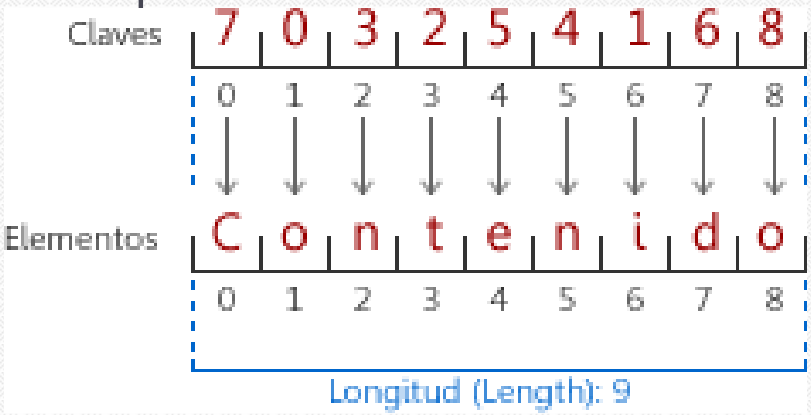
```
Array.Reverse(palabras);
// Salida: y – verde – salta – queda – mira – lindo – el – ejercitación
```

Ordenación

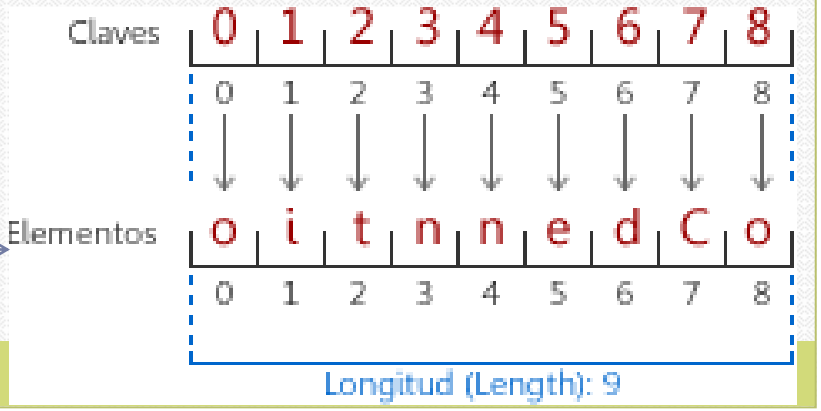
- **Ejercicio 1**
  - Crear un programa que permita al usuario ingresar N números y M letras (N puede ser distinto o igual a M). Armar un arreglo para los números ingresados y un arreglo para las letras y:
    - Mostrar los arreglos originales,
    - Mostrar los arreglos ordenados de menor a mayor,
    - Mostrar los arreglos de mayor a menor.

ORDENACIÓN DE ARRAY SIMULTÁNEOS

- **Sort(arregloClaves, arregloElementos)**
  - Ordena un par de objetos **Array** unidimensionales donde el primero de ellos contiene las claves y el segundo los elementos correspondientes. La ordenación se realiza en función del arreglo que contiene las claves. Por ejemplo:



Array.Sort(Claves, Elementos)



ORDENACIÓN

- **Ejercicio 2**
  - Se debe realizar un programa que le pida al usuario que indique cuántos alumnos cargará y luego le solicite el nombre y apellido del mismo, y la carrera que cursa. Se deben almacenar estos datos por separado en dos arreglos distintos, y luego el usuario podrá visualizarlos ordenados por nombre o por carrera, según elija. El sistema deberá repetir estas opciones tantas veces como el usuario desee (para lo cual deberá preguntarle si desea continuar)

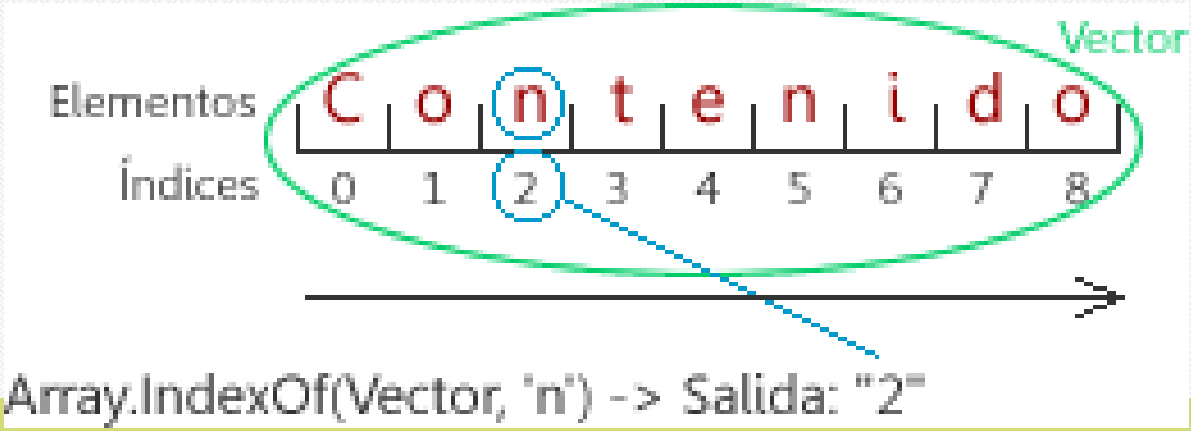
Clase Array

funcionalidad

BÚSQUEDA EN UN ARRAY

- **IndexOf**
  - Busca un objeto determinado en todo el arreglo o parte de éste, y devuelve el índice de la primera aparición del mismo. Por ejemplo:

```
string[] claves = { "rojo", "verde", "naranja", "azul", "lila", "naranja" };
int indice = Array.IndexOf(claves, "naranja"); // 2
Array.IndexOf(claves, "naranja", 3); // 5
Array.IndexOf(claves, "naranja", 0, 3); // 2
// Array.IndexOf(arreglo, objeto, posicionInicial, cantElementosBuscar)
```



Clase Array

funcionalidad

BÚSQUEDA EN UN ARRAY

- **LastIndexOf**
  - Devuelve el índice de la última aparición de un valor en una matriz unidimensional o en una parte de ésta. La búsqueda se realiza a partir del último elemento (o la posición indicada) hasta el inicio o lo especificado. Por ejemplo:

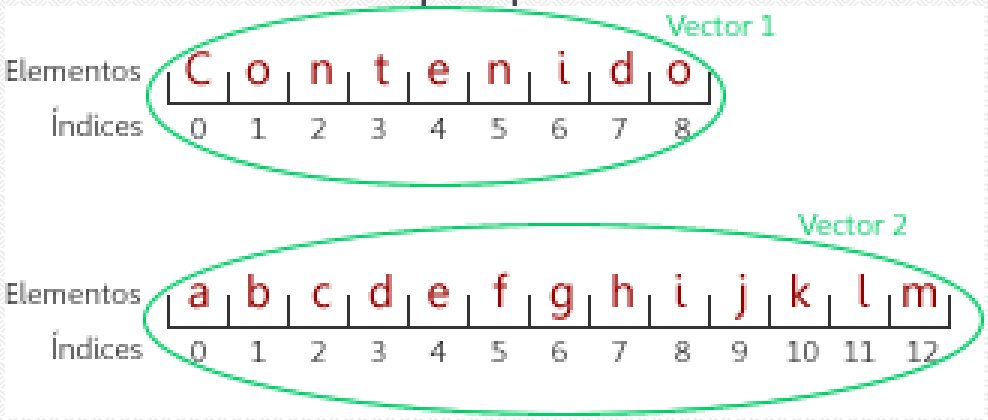
```
string[] claves = { "rojo", "verde", "naranja", "azul", "lila", "naranja" };
Array.LastIndexOf(claves, "naranja"); // Salida: 5
Array. LastIndexOf(claves, "naranja", 3); // Salida: 2
Array. LastIndexOf(claves, "naranja", 3, 3); // Salida: 2
// Array.IndexOf(arreglo, objeto, posicionInicial, cantElemHaciaAtrás)
```

Búsqueda en un Array

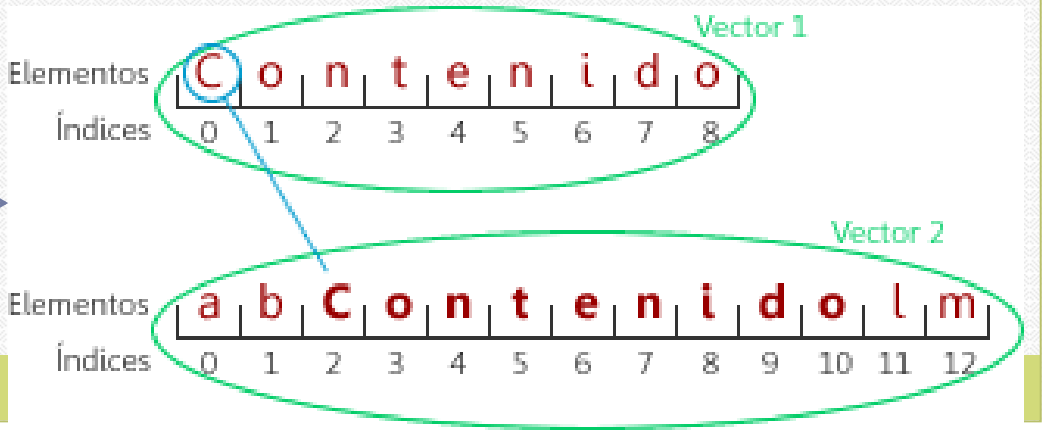
- **Ejercicio 3**
  - Se necesita construir un pequeño diccionario que permita al usuario ingresar una palabra en inglés y se muestre por pantalla su traducción/significado en español. El usuario debe poder buscar tantas palabras como desee (se le pregunta si desea continuar). Las palabras a incluir son, por lo menos:
    - Protocol, index, all, user, machine, window, last, abstract, source, device, mouse.
  - Además se le debe dar al usuario la oportunidad de listar todas las palabras que contiene el diccionario ordenadas alfabéticamente.

BÚSQUEDA EN UN ARRAY

- **CopyTo**
  - Copia todos los elementos del arreglo actual en el arreglo destino especificado, empezando en el índice indicado del arreglo de destino. Por ejemplo:



Vector1.CopyTo(Vector2, 2);



Búsqueda en un Array

- **Ejercicio 4**
  - Dado un arreglo de N números enteros, calcular y mostrar:
    - La suma de todos ellos,
    - La suma de los pares,
    - El promedio de todos los números,
    - El mayor,
    - El menor,
    - Todos los números ingresados ordenados de mayor a menor y de menor a mayor.

Búsqueda en un Array

- **Ejercicio 5**
  - Se parte de 3 arreglos distintos:
    - {"había", "una", "luz", "camino", "árbol"}
    - {"el", "álamo", "quiebre", "andén", "y", "justo"}
    - {"zona", "cuando", "antes"}
  - Y se necesita unificarlos y mostrarlos ordenadamente por pantalla.
  - Sugerencia: considere utilizar el método CopyTo.