UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

LABORATORIO - SEMANA 2 B

Integrantes:

Alumno	Código
Dominik Aldahir Mendoza Ramos	U20201B980
Ryan Robert Sweden Silva	U202011397
Nicole Areli Price Torrejón	U201921559
Ludwin Roy Reyes Suarez	U20201C339
Johan Raúl Moreno Vergara	U20201C105

Enlace: https://github.com/SoftwareSynth/upc-pre-laboratories/tree/main/lab-02-b

Documento EJER1.txt:

El código se encuentra funcional y, en consecuencia, reúne los requisitos para ser implementado en **producción**. No obstante, sería recomendable considerar la incorporación de prácticas óptimas a fin de elevar su calidad y legibilidad.

En este sentido, sugiero la adopción de un enfoque basado en el bucle **foreach** en lugar de la estructura '**for'** actualmente empleada para la iteración a través de la lista de usuarios. El uso del bucle foreach no solo conlleva mayor concisión, sino también una reducción de posibles errores.

```
foreach (User user in listOfUsers)
{
   Console.WriteLine($"{user.Name} is {user.Age} years old");
}
```

Documento EJER2.txt:

El código es funcional y cumple con el propósito de ordenar y mostrar una lista de usuarios por su edad.
Otro consejo que podríamos aplicar es utilizar nombres significativos para las funciones. En este caso en vez de “CompareUsers”, podríamos nombrar el método como “OrderByAge” para hacer su función más evidente.

Antes:

```
class Program
  {
  static void Main(string[] args)
  {
    List<User> listOfUsers = new List<User>()
    {
    new User() { Name = "John Doe", Age = 42 },
    new User() { Name = "Jane Doe", Age = 39 },
    new User() { Name = "Joe Doe", Age = 13 },
    };
    listOfUsers.Sort(CompareUsers);
    foreach (User user in listOfUsers)
    Console.WriteLine(user.Name + ": " + user.Age + " years old");
  }

  public static int CompareUsers(User user1, User user2)
  {
    return user1.Age.CompareTo(user2.Age);
  }
  }
```

Después:

```
class Program
  {
  static void Main(string[] args)
  {
    List<User> listOfUsers = new List<User>()
    {
    new User() { Name = "John Doe", Age = 42 },
    new User() { Name = "Jane Doe", Age = 39 },
    new User() { Name = "Joe Doe", Age = 13 },
    };
    listOfUsers.Sort(OrderByAge);
    foreach (User user in listOfUsers)
```

```
    Console.WriteLine(user.Name + ": " + user.Age + " years old");
}

public static int OrderByAge(User user1, User user2)
{
    return user1.Age.CompareTo(user2.Age);
}
}
```

Luego de estos cambios podemos asegurar que el código puede pasar a producción.

Documento EJER3.txt:

El código no presenta errores durante el proceso de ejecución. Por ende, puede ser implementado en **producción**. El código utiliza la clase ArrayList para trabajar con una colección de elementos. Esta es una clase válida para trabajar con colecciones en C#, desde .NET Framework 2.0 se recomienda utilizar List<T> genérica para obtener un mejor rendimiento y mayor seguridad en tiempo de compilación. Por ello, se sugiere el cambio de ArrayList por List<T>.

Documento EJER4.txt:

El código no presenta errores de compilación, como se muestra en la ejecución.

```
Primero Segundo Tercero Cuarto Quinto
La segunda cadena es: Segundo
Al principio
Primero
Después de Primero
Segundo
Tercero
Cuarto
Quinto
La palabra "Tercero " está en la posición 4
Contenido tras ordenar
Al principio
Cuarto
Después de Primero
Primero
Quinto
Segundo
Tercero
Ahora "Cuarto " está en la posición 1
Contenido dar la vuelta y tras eliminar dos:
Tercero
Primero
Después de Primero
Cuarto
Al principio
La frase "Primero " ...
Está en la posición 3
La frase "Sexto " ...
No está. El dato inmediatamente mayor es el 4: Tercero
```

Sin embargo, se identifacaron unas posibles mejoras funcionales:

- Se recomienda evitar el uso de List frente a ArrayList, ya que está obsoleto y no es genérico con ello se mejora la seguridad y rendimiento. Además, la lógica para buscar elementos en ArrayList y determinar su posición es confusa.

- En lugar de emplear bucles for junto a miArrayList.Count para recorrer la lista, es preferible utilizar bucles foreach. Esto aumenta la legibilidad y reduce posibles errores.

- Se detecta falta de claridad en las salidas de datos en distintas partes del código, lo que puede confundir a los usuarios. Es necesario mejorar la redacción de los mensajes y la presentación de los resultados.

Antes:

```
ArrayList miArrayList = new ArrayList();
...
for (int i = 0; i < miArrayList.Count; i++)
```

Después:

```
List<string> miLista = new List<string>();
...
foreach (string frase in miLista)
```

Evaluación Final para Producción:

Basándonos en las observaciones anteriores, es necesario corregir y mejorar el código antes de considerar su implementación en producción. La seguridad de tipos, el manejo de excepciones y la lógica confusa son problemas a resolver. Se sugieren ajustes y pruebas exhaustivas previas a la implementación en producción.

Conclusiones:

En resumen, la gran mayoría de los códigos analizados son funcionales y pueden ser implementados en producción. No obstante, se sugieren mejoras para optimizar la calidad y legibilidad del código, como el uso de bucles 'foreach' en lugar de 'for', nombres descriptivos para funciones y la actualización de las estructuras de datos a opciones más modernas y seguras, como cambiar `ArrayList` por `List<T>`. Estas sugerencias contribuirán a un código más eficiente y robusto en un entorno de producción.