Ejercicios de Listas: Lista Simple, Lista Circular y Lista Doblemente Enlazada

Información del Proyecto

Descripción	Detalles
Profesores	Sergio Cavero y Salvador Sanchez
Asignatura	Estructuras de Datos
Universidad	Universidad Rey Juan Carlos
Curso	2024/2025

Tabla de contenidos

¿Cómo enfrentarse a los ejercicios?

Antes de comenzar con cada uno de los ejercicios:

1. Lee detenidamente el enunciado **completo**.

2. Identifica el objetivo del ejercicio:

   - Identifica el tipo de elementos que se almacenará en la estructura de datos:
     - Ejemplos: enteros, caracteres, registros, etc.

        - Esto va a determinar la definición de los nodos de la misma (o el tipo de array en caso de no estar implementada en memoria dinámica).
    - Analiza si debes extender o modificar funcionalidades de un TAD existente:
        - Ejemplos: contar el número de elementos de una lista, verificar si un elemento está en la lista, eliminar un elemento, etc.
        - Generar nuevos procedimientos o funciones requerirá utilizar las operaciones principales.
        - Es decir, nuevamente deberemos **trabajar en la propia unidad** (TAD lista) ya que estamos extendiendo su funcionalidad. Tendremos en este caso el "rol" de implementador de la unidad.
    - Utilizar nuestro TAD para resolver un problema específico:
        - Ejemplos de TADs anteriores: verificar si una expresión aritmética está balanceada (TAD Pila), verificar si una palabra es palíndroma, utilizarla para representar un conjunto de elementos, etc.
        - En este caso, deberemos utilizar las operaciones básicas de la unidad que nos dan para resolver el problema, pero no podemos modificar el TAD lista.
        - Por lo tanto, deberemos **usar** la lista en el programa principal para resolver el problema. Tendremos el rol de usuario externo de la unidad y por lo tanto no la podremos modificar.

3. Abre cada uno de los archivos proporcionados y estudia el código que te damos.

4. Haz que el programa compile y ejecútalo, aunque de primeras no haga todo lo que se pide.

5. Realiza los ejercicios de manera ordenada, comprobando que cada uno de ellos funciona correctamente antes de pasar al siguiente.

Ejercicios con Listas Enlazadas Simples

Ejercicio 1: Cálculo de la complejidad en notación O grande

Calcula la complejidad en notación O grande de todas las operaciones de la lista enlazada simple vista en este tema.

Para ello, se proporciona el código de la lista enlazada simple en el archivo `uListaEnlazadaSimple.pas`.

Ejercicio 2: Instrucciones guiadas

En este ejercicio, trabajaremos con operaciones básicas de la lista enlazada simple.

- `uListaEnlazadaSimple.pas`: Implementación de la lista enlazada simple. Este fichero contiene la implementación de la lista enlazada simple que se utilizará en este ejercicio. No debes modificar este archivo.
- `listas_ej1.pas`: Programa principal que utiliza la lista enlazada simple. Este es el fichero en el que deberás realizar las implementaciones necesarias.

¿Qué rol tendremos en este ejercicio? Estaremos utilizando la lista enlazada simple para realizar operaciones básicas y por lo tanto, tendremos el "rol" de usuario del TAD, no de implementador. No

modificaremos la implementación de la lista enlazada simple, sino que utilizaremos las operaciones básicas que nos proporciona.

Realiza las siguientes operaciones sobre listas enlazadas y verifica el estado resultante:

1. Inicialización de la lista
    - Crear una lista vacía.
    - Estado esperado: `[]`
2. Inserción de elementos al final
    - Insertar en orden: `1`, `2`, `3`, `4`, `5` (siempre al final).
    - Estado esperado: `[1, 2, 3, 4, 5]`
3. Cálculo de diferencia entre primer y último elemento
    - Obtener el primer elemento (`1`) y el último elemento (`5`).
    - Calcular la diferencia: `primerElemento - ultimoElemento = -4`.
    - Estado esperado: La lista no cambia: `[1, 2, 3, 4, 5]`.
4. Eliminación y limpieza
    - Mostrar el número de elementos (tras implementar la parte (3) debería ser `5`).
    - Eliminar el primer elemento.
    - Estado esperado después de eliminar: `[2, 3, 4, 5]` (nuevo tamaño: `4`).
    - Limpiar la lista completamente.
    - Estado esperado: `[]` (nuevo tamaño: `0`).
5. Realizar una copia o duplicado de una lista y verificación de lista vacía
    - Verificar si la lista está vacía (debe ser `true`).
    - Insertar `1` al final.
    - Estado esperado: `[1]` (no vacía).
    - Crear una copia de la lista (`lista2`).
    - Eliminar el primer elemento de `lista2`.
    - Estado esperado:
        - `lista1`: `[1]`
        - `lista2`: `[]`.
6. Inserción al inicio
    - Insertar `0` al inicio de `lista1`.
    - Estado esperado: `[0, 1]`.
7. Eliminación de un elemento al final
    - Eliminar el último elemento de `lista1`.
    - Estado esperado: `[0]`.
8. Eliminación de un elemento específico

    - Insertar `3` al final.
    - Estado antes de eliminar: `[0, 3]`.
    - Eliminar el elemento `3`.
    - Estado esperado: `[0]`.

9. Búsqueda de un elemento (iterativa)

    - Verificar si el elemento `2` está en la lista.
    - Resultado esperado: `El 2 no está en la lista`.

10. Igual al inicio y al final

    - Verificar si el elemento del final de la lista es igual al del inicio. Si la lista está vacía consideraremos que esta condición también se cumple.
    - Resultado esperado: `Los elementos inicial y final son iguales (0)`.

Ejercicio 3: Añadir una nueva funcionalidad a la lista enlazada simple

En este ejercicio, añadiremos nueva funcionalidad a la lista enlazada simple. Concretamente, crearemos la función `set_at` que permitirá modificar el valor de un elemento en una posición específica de la lista. Consideraremos que las posiciones de los elementos son las siguientes: elemento al principio posición 1, siguiente elemento posición 2, etc.

Para realizar este ejercicio, se proporcionan los siguientes archivos:

- `uListaEnlazadaSimpleSetAt.pas`: Implementación de la lista enlazada simple. Este fichero contiene la implementación de la lista enlazada simple que se utilizará en este ejercicio. Deberás modificar este archivo para añadir la nueva funcionalidad.
- `listas_ej3.pas`: Programa principal que utiliza la lista enlazada simple. No es necesario modificar este archivo.

¿Qué rol debes asumir en este ejercicio? Dado que estamos añadiendo nueva funcionalidad a la lista enlazada simple, tu "rol" es el de implementador del TAD lista. Por lo tanto, deberemos modificar la unidad de la lista enlazada simple para añadir la nueva funcionalidad.

Ejemplo de uso de la función `set_at`:

1. Lista vacia `[ ]`

    - `set_at([], 1, 10)`
    - Salida esperada: `[]` (sin cambios)

2. Lista `[1, 2, 3]`

    - `set_at([1, 2, 3], 1, 10)`
    - Salida esperada: `[10, 2, 3]`

3. Lista `[1, 2, 3]`

    - `set_at([1, 2, 3], 2, 10)`
    - Salida esperada: `[1, 10, 3]`

4. Lista `[1, 2, 3]`
    - `set_at([1, 2, 3], 3, 10)`
    - Salida esperada: `[1, 2, 10]`
5. Lista `[1, 2, 3]`
    - `set_at([1, 2, 3], 4, 10)`
    - Salida esperada: `[1, 2, 3]` (sin cambios)
6. Lista `[1, 2, 3]`
    - `set_at([1, 2, 3], 0, 10)`
    - Salida esperada: `[1, 2, 3]` (sin cambios)

Al ejecutar el programa principal `listas_ej3.pas`, se deben mostrar los resultados de las pruebas realizadas. Por ejemplo:

```
Comprobación del método set_at en uListaEnlazadaSimple:

Test set_at: Lista vacía
  Resultado = , Esperado = , Test: bien.
Test set_at: Posición 1
  Resultado = 10 2 3 , Esperado = 10 2 3 , Test: bien.
Test set_at: Posición media
  Resultado = 1 10 3 , Esperado = 1 10 3 , Test: bien.
Test set_at: Posición final
  Resultado = 1 2 10 , Esperado = 1 2 10 , Test: bien.
Test 1 set_at: Posición fuera de rango
  Resultado = 1 2 3 , Esperado = 1 2 3 , Test: bien.
Test 2 set_at: Posición fuera de rango
  Resultado = 1 2 3 , Esperado = 1 2 3 , Test: bien.
```

Ejercicio 4: Dividir lista en pares e impares

En este ejercicio, implementaremos un subprograma que reciba una lista de enteros y devuelva dos listas: una con los números pares y otra con los impares. La lista original no debe ser modificada.

Para realizar este ejercicio, se proporcionan los siguientes archivos:

- `uListaEnlazadaSimple.pas`: Implementación de la lista enlazada simple. Este fichero contiene la implementación de la lista enlazada simple que se utilizará en este ejercicio. No es necesario modificar este archivo.
- `listas_ej4.pas`: Programa principal que utiliza la lista enlazada simple.

¿Qué rol tendremos en este ejercicio? Estaremos utilizando la lista enlazada simple para realizar operaciones básicas. Por lo tanto, tendremos el "rol" de usuario externo de la unidad. No modificaremos la implementación de la lista enlazada simple, sino que utilizaremos las operaciones básicas que nos proporciona.

Localiza en el archivo `listas_ej4.pas` la función `dividir_lista_en_pares_e_impares` y completa su implementación. Esta función debe recibir una lista de enteros y devolver dos listas: una con los números pares y otra con los impares. La lista original no debe ser modificada.

Ejemplos de uso de la función `dividir_lista_en_pares_e_impares`:

1. **Lista con números mixtos**:
    - Lista original: `[1, 2, 3, 4, 5, 6]`
    - Lista de pares esperada: `[2, 4, 6]`
    - Lista de impares esperada: `[1, 3, 5]`
2. **Lista vacía**:
    - Lista original: `[]`
    - Lista de pares esperada: `[]`
    - Lista de impares esperada: `[]`
3. **Lista con solo números pares**:
    - Lista original: `[2, 4, 6]`
    - Lista de pares esperada: `[2, 4, 6]`
    - Lista de impares esperada: `[]`
4. **Lista con solo números impares**:
    - Lista original: `[1, 3, 5]`
    - Lista de pares esperada: `[]`
    - Lista de impares esperada: `[1, 3, 5]`

Al ejecutar el programa principal `listas_ej4.pas`, se deben mostrar los resultados de las pruebas realizadas. Por ejemplo:

```
Ejercicio: Dividir lista en pares e impares
  Lista original: 1 2 3 4 5 6
  Test 1: Lista pares = 2 4 6 , Correcta = 2 4 6  El ejercicio funciona en
pares bien.
  Test 1: Lista impares = 1 3 5 , Correcta = 1 3 5  El ejercicio funciona
en impares bien.
  Estado lista inicial sin modificar: 1 2 3 4 5 6  = 1 2 3 4 5 6  bien.
  Lista original:
  Test 2 (Lista vacía): Lista pares = , Correcta =  El ejercicio funciona
en pares bien.
  Test 2 (Lista vacía): Lista impares = , Correcta =  El ejercicio
funciona en impares bien.
  Estado lista inicial sin modificar:  =  bien.
  Lista original: 2 4 6
  Test 3 (Solo pares): Lista pares = 2 4 6 , Correcta = 2 4 6  El
ejercicio funciona en pares bien.
  Test 3 (Solo pares): Lista impares = , Correcta =  El ejercicio funciona
en impares bien.
  Estado lista inicial sin modificar: 2 4 6  = 2 4 6  bien.
```

```
  Lista original: 1 3 5
  Test 4 (Solo impares): Lista pares = , Correcta =  El ejercicio funciona
en pares bien.
  Test 4 (Solo impares): Lista impares = 1 3 5 , Correcta = 1 3 5  El
ejercicio funciona en impares bien.
  Estado lista inicial sin modificar: 1 3 5  = 1 3 5  bien.
```

Ejercicio 5: Unir dos listas enlazadas simples

En este ejercicio, implementaremos un subprograma que reciba dos listas de enteros ordenadas y devuelva una nueva lista ordenada con los elementos de ambas listas. Las listas originales no deben ser modificadas.

Para realizar este ejercicio, se proporcionan los siguientes archivos:

- `uListaEnlazadaSimple.pas`: Implementación de la lista enlazada simple. Este fichero contiene la implementación de la lista enlazada simple que se utilizará en este ejercicio. No es necesario modificar este archivo.
- `listas_ej5.pas`: Programa principal que utiliza la lista enlazada simple.

¿Qué rol tendremos en este ejercicio? Estaremos utilizando la lista enlazada simple para realizar operaciones básicas. Por lo tanto, tendremos el "rol" de usuario externo de la unidad. No modificaremos la implementación de la lista enlazada simple, sino que utilizaremos las operaciones básicas que nos proporciona.

Ejemplo de uso del subprograma `unir_listas_ordenadas`:

1. **Listas sin elementos comunes**:
    - Lista 1: `[1, 3, 5]`
    - Lista 2: `[2, 4, 6]`
    - Lista resultante esperada: `[1, 2, 3, 4, 5, 6]`
2. **Listas con elementos comunes y diferentes longitudes**:
    - Lista 1: `[1, 2, 7, 9]`
    - Lista 2: `[3, 5, 8]`
    - Lista resultante esperada: `[1, 2, 3, 5, 7, 8, 9]`
3. **Una lista vacía y otra con elementos**:
    - Lista 1: `[]`
    - Lista 2: `[2, 4, 6]`
    - Lista resultante esperada: `[2, 4, 6]`
4. **Ambas listas vacías**:
    - Lista 1: `[]`
    - Lista 2: `[]`
    - Lista resultante esperada: `[]`
5. **Listas con algunos elementos duplicados**:

- Lista 1: `[1, 2, 7, 9]`
- Lista 2: `[1, 2, 7, 9]`
- Lista resultante esperada: `[1, 1, 2, 2, 7, 7, 9, 9]`

Al ejecutar el programa principal `listas_ej5.pas`, se deben mostrar los resultados de las pruebas realizadas. Por ejemplo:

```
Ejercicio: Unir listas ordenadas
  Listas a unir: lista1 = 1 3 5 , lista2 = 2 4 6
  Test 1: Resultado = 1 2 3 4 5 6 , Correcto = 1 2 3 4 5 6  El ejercicio
funciona bien.
  Estado lista inicial 1 sin modificar: bien.
  Estado lista inicial 2 sin modificar: bien.
  Listas a unir: lista1 = 1 2 7 9 , lista2 = 3 5 8
  Test 2: Resultado = 1 2 3 5 7 8 9 , Correcto = 1 2 3 5 7 8 9  El
ejercicio funciona bien.
  Estado lista inicial 1 sin modificar: bien.
  Estado lista inicial 2 sin modificar: bien.
  Listas a unir: lista1 = , lista2 = 2 4 6
  Test 3: Resultado = 2 4 6 , Correcto = 2 4 6  El ejercicio funciona
bien.
  Estado lista inicial 1 sin modificar: bien.
  Estado lista inicial 2 sin modificar: bien.
  Listas a unir: lista1 = , lista2 =
  Test 4: Resultado = , Correcto =  El ejercicio funciona bien.
  Estado lista inicial 1 sin modificar: bien.
  Estado lista inicial 2 sin modificar: bien.
  Listas a unir: lista1 = 1 2 7 9 , lista2 = 1 2 7 9
  Test 5: Resultado = 1 1 2 2 7 7 9 9 , Correcto = 1 1 2 2 7 7 9 9  El
ejercicio funciona bien.
  Estado lista inicial 1 sin modificar: bien.
  Estado lista inicial 2 sin modificar: bien.
```

Ejercicio 6: Modificar la unidad de lista enlazada simple

En este ejercicio, modificaremos la unidad de lista enlazada simple para incluir un array de frecuencias que cuente la cantidad de veces que cada entero entre `MIN_RANGE` y `MAX_RANGE` aparece en la lista. Para ello, se proporcionan los siguientes archivos:

- `uListaEnlazadaSimpleMod.pas`: Implementación de la lista enlazada simple que debe ser modificada. Este fichero contiene la implementación de la lista enlazada simple que se utilizará en este ejercicio. Deberás modificar este archivo para añadir la nueva funcionalidad.
- `listas_ej6.pas`: Programa principal que utiliza la lista enlazada simple modificada y comprueba su correcto funcionamiento. No debería ser necesario modificar este archivo.

Tener una tabla de frecuencias requerira definir una estructura de datos auxiliar que almacene la frecuencia de cada número. En este caso, utilizaremos un array de enteros para almacenar las frecuencias de los números en el rango que puede ser especificado por medio de constantes `MIN_RANGE` y `MAX_RANGE`.

Tener una tabla de frecuencias nos permitirá realizar operaciones como obtener la frecuencia de un número en tiempo constante (O(1)), verificar si un número está en la lista en tiempo constante (O(1)), etc.

¿Qué rol tendremos en este ejercicio? Estaremos añadiendo nueva funcionalidad a la lista enlazada simple, por lo que tendremos el "rol" de implementador de la unidad. Por lo tanto, deberemos modificar la unidad de la lista enlazada simple para añadir la nueva funcionalidad.

Antes de que sigas leyendo, te recomendamos que intentes resolver el ejercicio por tu cuenta. Si no puedes hacerlo, sigue los siguientes pasos:

1. **Definir constantes y tipos**:
    - Define las constantes `MIN_RANGE`, `MAX_RANGE` para establecer el rango de valores y el tamaño del array de frecuencias.
    - Define el tipo `tFrecuencias` como un array estático de enteros para almacenar las frecuencias.
    - Modifica la estructura `tListaSimpleMod` para incluir el array de frecuencias.
2. **Inicialización**:
    - Modifica el procedimiento `initialize` para inicializar el array de frecuencias a cero.
3. **Inserción**:
    - Modifica los procedimientos `insert_at_end` e `insert_at_begin` para incrementar la frecuencia del número insertado.
4. **Eliminación**:
    - Modifica los procedimientos `delete_at_end`, `delete_at_begin` y `delete` para decrementar la frecuencia del número eliminado.
5. **Búsqueda**:
    - Modifica la función `in_list` para utilizar el array de frecuencias y verificar si un número está en la lista en tiempo constante (O(1)).
6. **Frecuencia**:
    - Implementa la función `get_frequency` para obtener la frecuencia de un número en la lista.
    - Implementa la función `frequency_to_string` para obtener una representación en cadena del array de frecuencias.
7. **Limpieza y copia**:
    - Modifica el procedimiento `clear` para resetear también el array de frecuencias.
    - Modifica el procedimiento `copy` para copiar también el array de frecuencias.
8. **Pruebas**:
    - Realiza pruebas para verificar que las modificaciones funcionan correctamente. Para ello, ejecuta el archivo `listas_ej3.pas`.

Ejercicios con Listas Enlazadas Circulares

Ejercicio 7: Instrucciones guiadas - Listas Enlazadas Circulares

En este ejercicio, trabajaremos con operaciones básicas de la lista enlazada circular.

- `uListaEnlazadaCircular.pas`: Implementación de la lista enlazada circular. Este fichero contiene la implementación de la lista enlazada circular que se utilizará en este ejercicio. No es necesario modificar este archivo.
- `listas_ej7.pas`: Programa principal que utiliza la lista enlazada circular. Este es el fichero en el que deberás realizar las implementaciones necesarias.

¿Qué rol tendremos en este ejercicio? Estaremos utilizando la lista enlazada circular para realizar operaciones básicas. Por lo tanto, tendremos el "rol" de usuario externo de la unidad. No modificaremos la implementación de la lista enlazada circular, sino que utilizaremos las operaciones básicas que nos proporciona.

Realiza las siguientes operaciones sobre listas enlazadas circulares y verifica el estado resultante:

1. **Inicialización de la lista**

- Crear una lista vacía.
- Estado esperado: `[]`

2. **Inserción de elementos al final**

- Insertar en orden: `1`, `2`, `3`, `4`, `5` (siempre al final).
- Estado esperado: `[1 2 3 4 5 ]`

3. **Cálculo de diferencia entre primer y último elemento**

- Obtener el primer elemento (`1`) y el último elemento (`5`).
- Calcular la diferencia: `primerElemento - ultimoElemento = -4`.
- Estado esperado: La lista no cambia: `[1 2 3 4 5 ]`.

4. **Eliminación y limpieza**

- Mostrar el número de elementos (debe ser `5`).
- Eliminar el primer elemento.
- Estado esperado después de eliminar: `[2 3 4 5 ]` (nuevo tamaño: `4`).
- Limpiar la lista completamente.
- Estado esperado: `[]` (nuevo tamaño: `0`).

5. **Copia de lista y verificación de vacío**

- Verificar si la lista está vacía (debe ser `true`).
- Insertar `1` al final.
- Estado esperado: `[1 ]` (no vacía).
- Crear una copia de la lista (`lista2`).
- Eliminar el primer elemento de `lista2`.

- Estado esperado:
  - `lista1`: `[1 ]`
  - `lista2`: `[]`.

6. **Inserción al inicio**

- Insertar `0` al inicio de `lista1`.
- Estado esperado: `[0 1 ]`.

7. **Eliminación al final (Doble eliminación)**

- Eliminar el último elemento de `lista1`.
- Eliminar el último elemento de `lista1` nuevamente.
- Estado esperado: `[]`.

8. **Eliminación de elemento específico (con inserciones adicionales)**

- Insertar `3` al final de `lista1`.
- Insertar `0` al inicio de `lista1`.
- Estado antes de eliminar: `[0 3 ]`.
- Eliminar el elemento `3`.
- Estado esperado: `[0 ]`.

9. **Búsqueda de elemento (iterativa)**

- Verificar si el elemento `2` está en la lista.
- Resultado esperado: `El 2 está en la lista`

10. **Generación de una lista con n elementos aleatorios **

- Crea una lista nueva y genera 10 enteros aleatorios entre 0 y 100. Introdúcelos en la lista por el final.
- Resultado posible: `[13 51 23 24 65 71 30 98 11 8]`

Ejercicio 8: Desarrollar funcionalidades de la unidad de lista enlazada circular con recursividad

En este ejercicio, añadiremos nuevas funcionalidades a la lista enlazada circular utilizando recursividad. Concretamente, crearemos las funciones `to_string_rec` y `num_elems_rec` que permitirán obtener una representación en cadena de la lista y contar el número de elementos de la lista de manera recursiva respectivamente.

Para realizar este ejercicio, se proporcionan los siguientes archivos:

- `uListaEnlazadaCircularMod.pas`: Implementación de la lista enlazada circular. Este fichero contiene la implementación de la lista enlazada circular que se utilizará en este ejercicio. Deberás modificar este archivo para añadir las nuevas funcionalidades. Esta unidad es una versión idéntica a la de la lista enlazada circular, pero con el nombre modificado para evitar conflictos con la unidad original.
- `listas_ej8.pas`: Programa principal que utiliza la lista enlazada circular. No es necesario modificar este archivo.

¿Qué rol tendremos en este ejercicio? Estamos añadiendo nueva funcionalidad a la lista enlazada circular, por lo que tendremos el "rol" de implementador de la unidad. Por lo tanto, deberemos modificar la unidad de la lista enlazada circular para añadir las nuevas funcionalidades.

Función to_string_recc`

La función `to_string_rec` debe devolver una representación en cadena de la lista enlazada circular utilizando recursividad.

Ejemplo de uso de la función `to_string_rec`:

1. Lista vacía `[]`
    - `to_string_rec([])`
    - Salida esperada: `''` (cadena vacía)
2. Lista `[1, 2, 3]`
    - `to_string_rec([1, 2, 3])`
    - Salida esperada: `'1 2 3 '`

Función num_elems_recc`

La función `num_elems_rec` debe devolver el número de elementos de la lista enlazada circular utilizando recursividad.

Ejemplo de uso de la función `num_elems_rec`:

1. Lista vacía `[]`
    - `num_elems_rec([])`
    - Salida esperada: `0`
2. Lista `[1, 2, 3]`
    - `num_elems_rec([1, 2, 3])`
    - Salida esperada: `3`

Al ejecutar el programa principal `listas_ej8.pas`, se deben mostrar los resultados de las pruebas realizadas. Por ejemplo:

```
Ejercicio: Comparación num_elems iterativo vs recursivo
  Test 1: Lista con 5 elementos.
    num_elems (iterativo): 5
    num_elems_rec (recursivo): 5
    ¿Resultados iguales?: bien.
  Test 2: Lista con 2 elementos.
    num_elems (iterativo): 2
    num_elems_rec (recursivo): 2
    ¿Resultados iguales?: bien.
  Test 3: Lista con elementos repetidos (3 elementos).
    num_elems (iterativo): 3
```

```
    num_elems_rec (recursivo): 3
    ¿Resultados iguales?: bien.
  Test 4: Lista vacía.
    num_elems (iterativo): 0
    num_elems_rec (recursivo): 0
    ¿Resultados iguales?: bien.
  Test 5: Lista con un solo elemento.
    num_elems (iterativo): 1
    num_elems_rec (recursivo): 1
    ¿Resultados iguales?: bien.
  Test 6: Lista vacía después de clear().
    num_elems (iterativo): 0
    num_elems_rec (recursivo): 0
    ¿Resultados iguales?: bien.


Ejercicio: Comparación to_string iterativo vs recursivo
  Test 1: Lista con 5 elementos.
    to_string (iterativo): "1 2 3 4 5 "
    to_string_rec (recursivo): "1 2 3 4 5 "
    ¿Resultados iguales?: bien.
  Test 2: Lista con 2 elementos.
    to_string (iterativo): "10 20 "
    to_string_rec (recursivo): "10 20 "
    ¿Resultados iguales?: bien.
  Test 3: Lista con elementos repetidos (3 elementos).
    to_string (iterativo): "7 7 7 "
    to_string_rec (recursivo): "7 7 7 "
    ¿Resultados iguales?: bien.
  Test 4: Lista vacía.
    to_string (iterativo): ""
    to_string_rec (recursivo): ""
    ¿Resultados iguales?: bien.
  Test 5: Lista con un solo elemento.
    to_string (iterativo): "1 "
    to_string_rec (recursivo): "1 "
    ¿Resultados iguales?: bien.
  Test 6: Lista vacía después de clear().
    to_string (iterativo): ""
    to_string_rec (recursivo): ""
    ¿Resultados iguales?: bien.
```

Ejercicio 9: Eliminar los duplicados de una lista enlazada circular

En este ejercicio, implementaremos un subprograma que elimine los elementos duplicados de una lista enlazada circular. La lista original debe ser modificada para que contenga solo elementos únicos.

Para realizar este ejercicio, se proporcionan los siguientes archivos:

- `uListaEnlazadaCircular.pas`: Implementación de la lista enlazada circular. Este fichero contiene la implementación de la lista enlazada circular que se utilizará en este ejercicio. No es necesario modificar este archivo.

- `listas_ej9.pas`: Programa principal que utiliza la lista enlazada circular. Aquí deberás completar la implementación del procedimiento `eliminar_duplicados`.

¿Qué rol tendremos en este ejercicio? Estaremos utilizando la lista enlazada circular para realizar operaciones básicas. Por lo tanto, tendremos el "rol" de usuario externo de la unidad. No modificaremos la implementación de la lista enlazada circular, sino que utilizaremos las operaciones básicas que nos proporciona.

Localiza en el archivo `listas_ej9.pas` el procedimiento `eliminar_duplicados` y completa su implementación. Este procedimiento debe recibir una lista enlazada circular y eliminar todos los elementos duplicados, dejando solo una instancia de cada elemento. Concretamente, si un elemento aparece más de una vez en la lista, todas las instancias adicionales deben ser eliminadas, manteniendo única la primera aparición.

Ejemplos de uso del procedimiento `eliminar_duplicados`:

1. **Lista con duplicados múltiples**:
    - Lista original: `[2, 4, 1, 2, 3, 4, 4, 5]`
    - Lista resultante esperada: `[2, 4, 1, 3, 5]`
2. **Lista sin duplicados**:
    - Lista original: `[1, 2, 3, 4, 5]`
    - Lista resultante esperada: `[1, 2, 3, 4, 5]`
3. **Lista con todos los elementos duplicados**:
    - Lista original: `[2, 2, 2]`
    - Lista resultante esperada: `[2]`
4. **Lista con un solo elemento (sin duplicados)**:
    - Lista original: `[7]`
    - Lista resultante esperada: `[7]`
5. **Lista vacía**:
    - Lista original: `[]`
    - Lista resultante esperada: `[]`
6. **Lista con todos los elementos repetidos múltiples veces**:
    - Lista original: `[1, 1, 1, 1, 1]`
    - Lista resultante esperada: `[1]`

Al ejecutar el programa principal `listas_ej9.pas`, se deben mostrar los resultados de las pruebas realizadas. Por ejemplo:

```
Ejercicio: Eliminar Duplicados de Lista Circular
Resultado Test 1: 2 4 1 3 5
  Test 1: Lista original: 2 4 1 2 3 4 4 5 , Resultado esperado: 2 4 1 3 5
```

```
-> bien.
Resultado Test 2: 1 2 3 4 5
  Test 2: Lista original: 1 2 3 4 5 , Resultado esperado: 1 2 3 4 5  ->
bien.
Resultado Test 3: 2
  Test 3: Lista original: 2 2 2 , Resultado esperado: 2  -> bien.
Resultado Test 4: 7
  Test 4: Lista original: 7 , Resultado esperado: 7  -> bien.
Resultado Test 5:
  Test 5: Lista original: , Resultado esperado:  -> bien.
Resultado Test 6: 1
  Test 6: Lista original: 1 1 1 1 1 , Resultado esperado: 1  -> bien.
```