Unidad 2 – Listas

Ejercicio N°1

Implementar una función llamada “printLista” que reciba una lista y una posición “p” e imprima los elementos de esa lista a partir de la posición “p”. No debe modificarse la lista original. Ej: listaOriginal: 1->2->3->4->5->6->7->8->9->10 con p=5 Por pantalla se verá: 7->8->9->10

Ejercicio N°2

Implementar una función que recibe una lista de números enteros y un número entero “n” y que modifique la lista borrando todos los elementos de la lista que tengan el número “n”. Imprimir la lista antes de hacer el llamado a la función y luego del llamado a la función para mostrar que la lista original cambió. ej: lista: 1->2->3->4->5->3->7->8->3->10 con n=3 debe pasar con la función a 1->2->4->5->7->8->10.

Ejercicio N°3

Implementar una función que intercambie los elementos entre 2 listas de números enteros que recibe como parámetros. Si los tamaños de las listas son distintos, igual debe cambiar los datos y cambiarían los tamaños de cada lista. Ejemplo sea **lista1**= 15->1->8->35->40->25->12 y **lista2**= 3->4->912->45->66 al ejecutar la función quedarían **lista1**= 3->4->912->45->66 y **lista2**=15->1->8->35->40->25->12.

Ejercicio N°4

Escribir un programa que pida al usuario una palabra o frase y la almacene en una Lista separando letra por letra, luego pedirá al usuario una vocal que desee contar y, por último, se debe imprimir por pantalla la lista y el número de veces que aparece la vocal en la palabra o frase. Validar que la Lista no esté vacía y que la letra a contar que introduzca el usuario sea una vocal.

Ejercicio N°5

Crear un programa que pida al usuario dar elementos a una Lista de números enteros. Luego cree una función que reciba una lista int L1 y devuelva otra lista int L2 conteniendo los elementos repetidos de L1. Por ejemplo, si L1 almacena los valores 5->2->7-> 2->5->5->1, debe construirse una lista L2 con los valores 5->2. Si en L1 no hay elementos repetidos se debe indicar al usuario que no hay elementos repetidos en L1, de lo contrario imprimir ambas listas.

Ejercicio N°6:

Crea un programa que gestione una lista circular de contactos. Cada contacto contiene un nombre (cadena de caracteres) y un número de teléfono (cadena de caracteres).

Tareas

1. Agregar Contactos:
    - Usa los métodos de `CircList` para agregar varios contactos a la lista. Deberás insertar al menos cinco contactos con nombres y números de teléfono distintos.
2. Mostrar Contactos:
    - Implementa una función que recorra la lista circular y muestre todos los contactos en orden. Usa el método `imprimir()` de `CircList`.
3. Buscar un Contacto por Nombre:
    - Implementa una función que busque un contacto por su nombre. Usa el método `getDato(int pos)` para acceder a los datos de los nodos y busca el contacto en la lista.
4. Eliminar un Contacto por Nombre:
    - Implementa una función que elimine un contacto por su nombre. Deberás usar los métodos `getDato(int pos)` y `eliminarPorValor(T valor)` (supón que `eliminarPorValor` está implementado, aunque no esté explícito en el código proporcionado).
5. Contar Contactos:
    - Usa el método `getTamanio()` para contar el número de contactos en la lista y muestra el resultado.

Ejercicio N°7

Utiliza la clase `ListaDoble` para gestionar una lista de estudiantes, donde cada estudiante tiene un nombre (cadena de caracteres) y una edad (entero). Implementa las siguientes funcionalidades:

1. **Agregar Estudiantes**:
    - Usa los métodos de `ListaDoble` para agregar varios estudiantes a la lista. Debes insertar al menos cinco estudiantes con nombres y edades diferentes.
2. **Mostrar Estudiantes**:
    - Implementa una función que recorra la lista y muestre todos los estudiantes en orden. Utiliza el método `imprimir()` de `ListaDoble`.
3. **Buscar un Estudiante por Nombre**:
    - Implementa una función que busque un estudiante por su nombre. Utiliza el método `getDato(int pos)` para acceder a los datos de los nodos y busca al estudiante en la lista.
4. **Eliminar un Estudiante por Nombre**:
    - Implementa una función que elimine un estudiante por su nombre. Utiliza los métodos `getDato(int pos)` y `remover(int pos)` para encontrar y eliminar al estudiante de la lista.
5. **Contar Estudiantes**:
    - Usa el método `getTamanio()` para contar el número de estudiantes en la lista y muestra el resultado.

Ejercicio N°8: APLICACIÓN DE CIRCULARIDAD

Sistema de Monitor Publicitario Circular

Desarrolla un programa para gestionar un monitor publicitario que debe mostrar frases de anuncios de manera circular. Para lograr esto, implementarás una solución utilizando una lista circular simplemente enlazada.

Requisitos:

1. **Frases de Anuncios:**
    - Cada frase de anuncio será un texto simple que el monitor debe mostrar por vez.
2. **Mostrar Circularmente:**
    - Las frases deben mostrarse en un formato circular, es decir, una vez que se ha mostrado la última frase, el monitor debe volver a mostrar la primera y así infinitamente.
3. **Implementación:**
    - Utiliza una lista circular simplemente enlazada para gestionar el almacenamiento y la visualización de las frases. La lista circular debe permitir el recorrido continuo de los elementos.
4. **Operaciones Básicas:**
    - **Agregar Frases:** Capacidad para añadir nuevas frases a la lista circular.
    - **Eliminar Frases:** Capacidad para eliminar frases específicas de la lista.
    - **Mostrar Frases:** Implementa una función para mostrar las frases en el monitor de manera continua, recorriendo circularmente la lista e infinitamente.

Ejercicio N°9: APLICACIÓN DE DOBLE ENLAZADA.

Gestión de Historial de Navegación en un Navegador

Desarrolla un programa que simule el historial de páginas web visitadas en un navegador utilizando una lista doblemente enlazada. El programa debe permitir al usuario retroceder y avanzar entre las páginas visitadas, mostrando sus direcciones en pantalla.

Requisitos:

1. **Estructura de Datos:**
    - Usa el concepto de una lista doblemente enlazada para mantener el historial de páginas web visitadas.
    - Cada nodo de la lista debe almacenar la URL de la página web visitada.
2. **Funciones del Programa:**
    - Añadir Página: Permite agregar una nueva URL al final del historial de navegación.
    - Mostrar Histórico: Muestra todas las URLs del historial desde la más antigua a la más reciente.
    - Retroceder: Permite al usuario retroceder a la página web visitada anteriormente, mostrando la URL en pantalla.
    - Avanzar: Permite al usuario avanzar a la siguiente página web en el historial, mostrando la URL en pantalla.
3. **Interfaz del Usuario:**
    - La interfaz debe ser de línea de comandos y debe permitir al usuario ingresar URLs y navegar a través del historial usando comandos simples.

- Implementa un menú para que el usuario pueda elegir entre las opciones disponibles: añadir una página, mostrar el historial, retroceder, avanzar, y salir del programa.