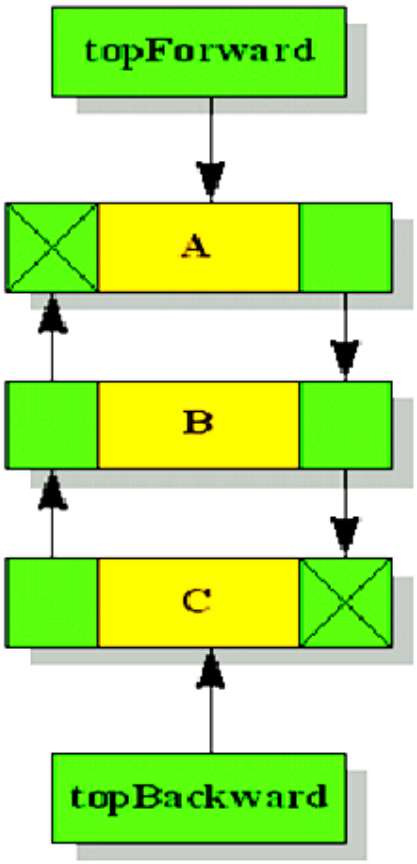
Algoritmos y Estructura de Datos

Práctica 7:
LISTAS DOBLEMENTE ENLAZADAS



Grupo: **2CM2**

Profra. Cecilia Alborante Morato

Integrantes:

- Guevara Badillo Areli Alejandra
- Hernández Simón Rebeca

30 de Mayo, 2023

INTRODUCCIÓN

Las listas doblemente enlazadas son una estructura de datos altamente versátil y eficiente utilizada en programación y ciencias de la computación. Representan una forma poderosa de almacenar y organizar datos de manera dinámica, ofreciendo una flexibilidad y un rendimiento superiores en comparación con las listas enlazadas simples. En esta introducción, exploraremos en profundidad qué son las listas doblemente enlazadas, cómo se diferencian de las listas enlazadas simples, por qué se consideran una mejora y cuáles son sus aplicaciones comunes.

Una lista doblemente enlazada es una estructura de datos lineal compuesta por nodos enlazados, donde cada nodo contiene una referencia al nodo anterior y al siguiente en la lista. Cada nodo también almacena un elemento de datos, lo que permite la construcción de colecciones de información organizadas. La capacidad de acceso bidireccional, es decir, acceder tanto al nodo anterior como al siguiente, es una característica clave que distingue a las listas doblemente enlazadas de las listas enlazadas simples.

La diferencia fundamental entre las listas doblemente enlazadas y las listas enlazadas simples radica en la dirección en la que se puede recorrer la lista. En una lista enlazada simple, solo se puede acceder a los nodos en una dirección, generalmente de principio a fin. En cambio, las listas doblemente enlazadas permiten recorrer la lista en ambas direcciones, lo que brinda una mayor flexibilidad y eficiencia en operaciones de inserción, eliminación y búsqueda.

Una de las ventajas clave de las listas doblemente enlazadas es su eficiencia para insertar y eliminar elementos en posiciones específicas de la lista. En lugar de tener que recorrer toda la lista desde el principio o realizar búsquedas exhaustivas, se puede acceder directamente al nodo deseado y ajustar los enlaces para insertar o eliminar el nodo de manera eficiente. Esto ahorra tiempo y recursos en comparación con las listas enlazadas simples, donde la búsqueda lineal es la única opción.

Además, las listas doblemente enlazadas son especialmente útiles cuando se requiere un acceso rápido a los elementos tanto en orden ascendente como descendente. Esto puede ser crucial en muchas aplicaciones, como sistemas de gestión de archivos, editores de texto y procesamiento de imágenes, donde se necesita un acceso eficiente y rápido a los elementos en función de su posición relativa. La capacidad de navegación bidireccional de las listas doblemente enlazadas simplifica enormemente estas operaciones y permite un procesamiento eficiente de los datos.

En resumen, las listas doblemente enlazadas ofrecen una solución más potente y flexible que las listas enlazadas simples al permitir el acceso bidireccional a los elementos y optimizar operaciones clave como inserciones, eliminaciones y búsquedas. Su eficiencia y capacidad de navegación bidireccional las convierten en una opción ideal en una amplia gama de aplicaciones, desde estructuras de datos complejas hasta sistemas de gestión de información. Al aprovechar las ventajas de las listas doblemente enlazadas, los programadores pueden desarrollar soluciones más eficientes y optimizadas en términos de rendimiento y tiempo de ejecución.

Planteamiento del Problema:

Implementa un programa que te permita realizar operaciones en una lista de datos de alumnos, como ingresar datos de alumnos, mostrar la lista de alumnos en forma ascendente o descendente, y modificar los datos de un alumno existente, por medio de listas doblemente enlazadas.

DESARROLLO

El programa presentado es una práctica 7, donde se trabaja con listas doblemente enlazadas para gestionar un registro de alumnos.

- Cada alumno se representa mediante una estructura que contiene información como la boleta, el nombre y los apellidos. Además, se utiliza un par de punteros, p y q, para apuntar al principio y al final de la lista respectivamente.

```
typedef struct Alumno {
    int boleta;
    char nombre[50];
    char apellidos[50];
    struct Alumno* siguiente;
    struct Alumno* anterior;
} Alumno;

Alumno* p = NULL;  // Puntero al primer nodo de la lista
Alumno* q = NULL;  // Puntero al último nodo de la lista
```

- La función *menu()* se encarga de presentar un menú interactivo al usuario, donde puede seleccionar diferentes opciones para interactuar con la lista de alumnos. El usuario puede agregar alumnos, mostrar la lista en orden ascendente o descendente, modificar los datos de un alumno o salir del programa. Se utiliza un bucle do-while para mantener el menú en ejecución hasta que el usuario decida salir.

```
void menu() {
    int opcion;

    printf("----- Menu -----\n");
    printf("1. Altas de alumnos\n");
    printf("2. Mostrar alumnos dados de alta\n");
    printf("3. Modificaciones de alumnos\n");
    printf("0. Salir\n");
    printf("Ingrese su opcion: ");

    do {
        scanf("%d", &opcion);
        system("pause");
        system("cls");

        switch (opcion) {
            case 1:
                datosAlumno();
                break;

            case 2:
                printf("2. Mostrar Listado\n--------------------------------\n");
                printf("\n1. Listado de alumnos en forma ascendente\n");
                printf("2. Listado de alumnos en forma descendente\n");

                printf("Ingrese su opcion: ");
                scanf("%d", &opcion);
                system("pause");
                system("cls");
```

```
                switch (opcion) {
                    case 1:
                        mostrarAscendente();
                        break;
                    case 2:
                        mostrarDescendente();
                        break;
                    default:
                        printf("Opción inválida.\n");
                }
                break;

            case 3:
                modificarAlumno();
                break;
            case 0:
                printf("Saliendo del programa...\n");
                exit(0);
                break;
            default:
                printf("Opción inválida.\n");
        }

        printf("\n");
    } while (opcion != 0);
}
```

- En la función *insertarAlumno()*, se realiza la inserción de un nuevo alumno en la lista. Se crea un nuevo nodo utilizando la función malloc() para asignar memoria dinámica, se asignan los valores correspondientes y se actualizan los punteros para mantener la coherencia de la lista. Dependiendo de la posición del nuevo alumno en relación a los demás, se ajustan los punteros anterior y siguiente de los nodos adyacentes.

```
void datosAlumno() {
    printf("1. Ingresar Alumno\n--------------------------------\n");
    int boleta;
    char nombre[50];
    char apellidos[50];

    printf("Ingrese la boleta del alumno: ");
    scanf("%d", &boleta);

    printf("Ingrese el nombre del alumno: ");
    scanf("%s", nombre);

    printf("Ingrese los apellidos del alumno: ");
    scanf("%s", apellidos);

    insertarAlumno(boleta, nombre, apellidos);
}
```

```
void insertarAlumno(int boleta, char* nombre, char* apellidos) {
    Alumno* nuevoAlumno = (Alumno*)malloc(sizeof(Alumno));

    nuevoAlumno->boleta = boleta;
    strcpy(nuevoAlumno->nombre, nombre);
    strcpy(nuevoAlumno->apellidos, apellidos);

    nuevoAlumno->siguiente = NULL;
    nuevoAlumno->anterior = NULL;

    if (p == NULL) {
        // Si la lista está vacía, el nuevo alumno será el primer y último nodo
        p = nuevoAlumno;
        q = nuevoAlumno;
    } else {
        Alumno* actual = p;
        Alumno* anterior = NULL;

        // Buscar la posición correcta en la lista ordenada por apellidos
        while (actual != NULL && strcmp(apellidos, actual->apellidos) > 0) {
            anterior = actual;
            actual = actual->siguiente;
        }

        if (actual != NULL) {
            if (anterior != NULL) {
                // Insertar el nuevo alumno en el medio de la lista
                anterior->siguiente = nuevoAlumno;
```

```
                nuevoAlumno->anterior = anterior;
                nuevoAlumno->siguiente = actual;
                actual->anterior = nuevoAlumno;
            } else {
                // Insertar el nuevo alumno al inicio de la lista
                nuevoAlumno->siguiente = p;
                p->anterior = nuevoAlumno;
                p = nuevoAlumno;
            }
        } else {
            // Insertar el nuevo alumno al final de la lista
            anterior->siguiente = nuevoAlumno;
            nuevoAlumno->anterior = anterior;
            q = nuevoAlumno;
        }
    }

    printf("\nAlumno registrado exitosamente :D\n");
    system("pause");
    system("cls");
    menu();
}
```

- La función *mostrarAscendente()* y *mostrarDescendente()* permiten mostrar la lista de alumnos en orden ascendente y descendente respectivamente. Estas funciones recorren la lista utilizando los punteros p y q y muestran los datos de cada alumno. Si la lista está vacía, se muestra un mensaje indicando que no hay alumnos registrados.

```
void mostrarAscendente() {
    printf("2. Mostrar Lista\n---------------------------------");
    printf(">> Listado Ascendente\n\n");
    Alumno* actual = p;

    if (actual == NULL) {
        printf("No hay alumnos registrados.\n");
        return;
    }

    printf("Listado de alumnos en forma ascendente:\n");
    while (actual != NULL) {
        printf("Boleta: %d\nNombre: %s\nApellidos: %s\n\n", actual->boleta, actual->nombre, actual->apellidos);
        actual = actual->siguiente;
    }

    system("pause");
    system("cls");
    menu();
}
```

```
void mostrarDescendente() {
    printf("2. Mostrar Lista\n---------------------------------");
    printf(">> Listado Descendente\n\n");

    Alumno* actual = q;

    if (actual == NULL) {
        printf("No hay alumnos registrados.\n");
        return;
    }

    printf("Listado de alumnos en forma descendente:\n");
    while (actual != NULL) {
        printf("Boleta: %d\nNombre: %s\nApellidos: %s\n\n", actual->boleta, actual->nombre, actual->apellidos);
        actual = actual->anterior;
    }

    system("pause");
    system("cls");
    menu();
}
```

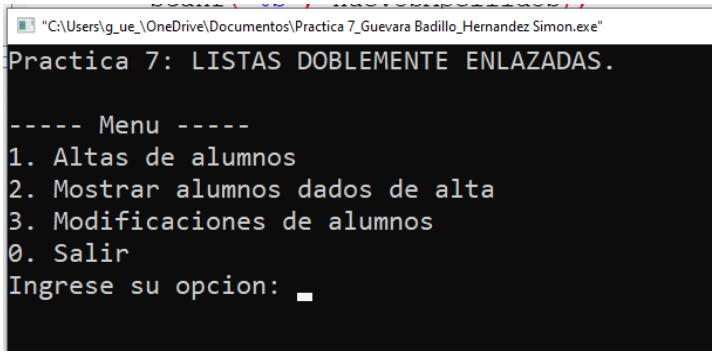
- La función *modificarAlumno()* permite realizar modificaciones en los datos de un alumno específico. Se solicita al usuario la boleta del alumno a modificar y se realiza una búsqueda en la lista utilizando un bucle. Si se encuentra el alumno, se solicitan los nuevos datos y se procede a eliminar el nodo actual para luego insertar uno nuevo con los datos actualizados.

```
void modificarAlumno() {
    printf("3. Modificar Alumno\n---------------------------------\n");

    Alumno* actual = p;
    int boleta;

    printf("Ingrese la boleta del alumno a modificar: ");
    scanf("%d", &boleta);

    while (actual != NULL && actual->boleta != boleta) {
        actual = actual->siguiente;
    }

    if (actual == NULL) {
        printf("No se encontró el alumno con la boleta especificada.\n");
    } else {
        char nuevoNombre[50];
        char nuevosApellidos[50];
        int boleta = actual->boleta;

        printf("Ingrese el nuevo nombre del alumno: ");
        scanf("%s", nuevoNombre);
        printf("Ingrese los nuevos apellidos del alumno: ");
        scanf("%s", nuevosApellidos);
```

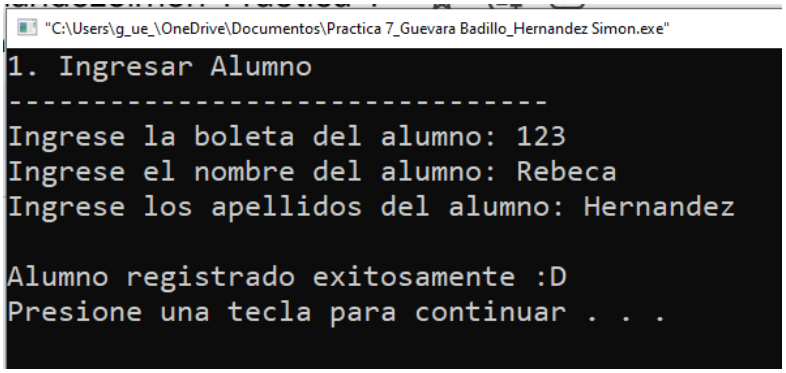
```
        if (actual->anterior == NULL) {
            // El alumno a modificar es el primer nodo de la lista
            p = actual->siguiente;
            free(actual);
        } else if (actual->siguiente == NULL) {
            // El alumno a modificar es el último nodo de la lista
            q = actual->anterior;
            free(actual);
        } else {
            // El alumno a modificar está en medio de la lista
            Alumno* anterior = actual->anterior;
            Alumno* siguiente = actual->siguiente;

            anterior->siguiente = siguiente;
            siguiente->anterior = anterior;
            free(actual);
        }

        insertarAlumno(boleta, nuevoNombre, nuevosApellidos);

        printf("Alumno modificado correctamente :D.\n");
    }

    system("pause");
    system("cls");
    menu();
}
```

RESULTADOS

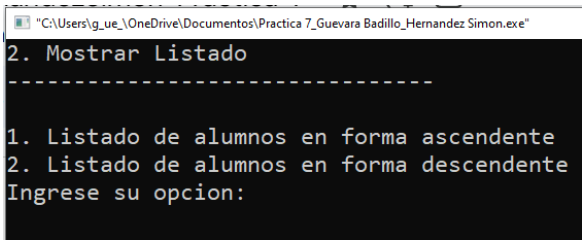
Menu:



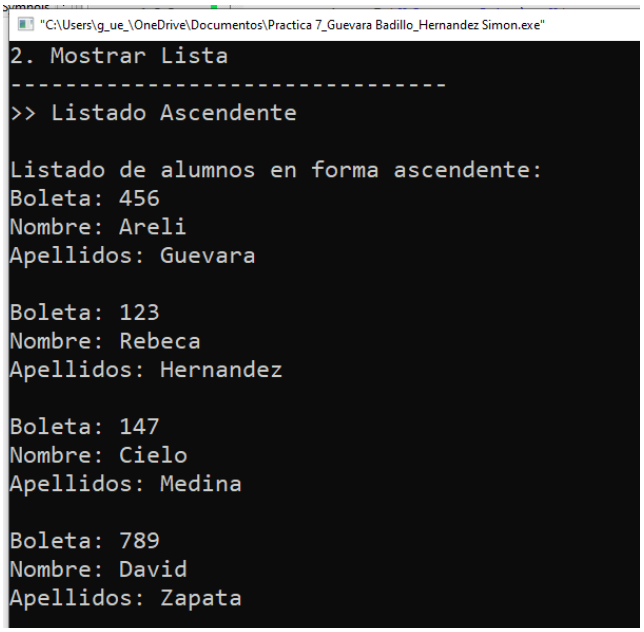
Insertar:



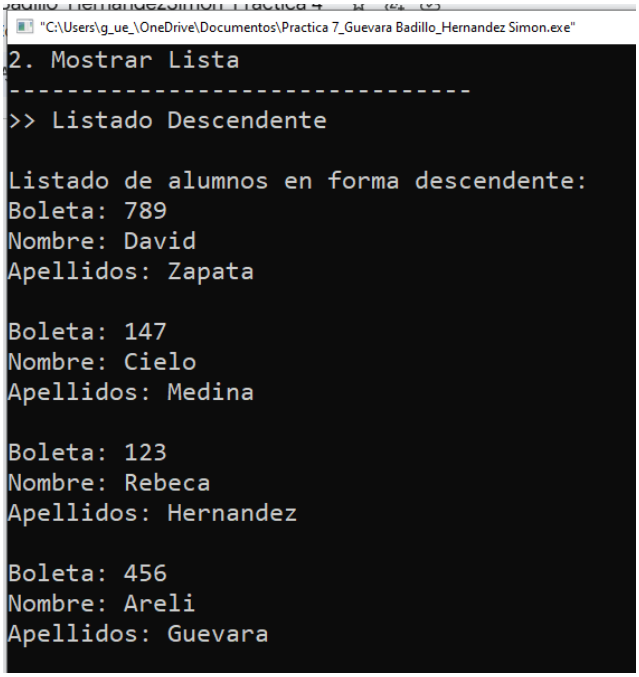
Mostrar:



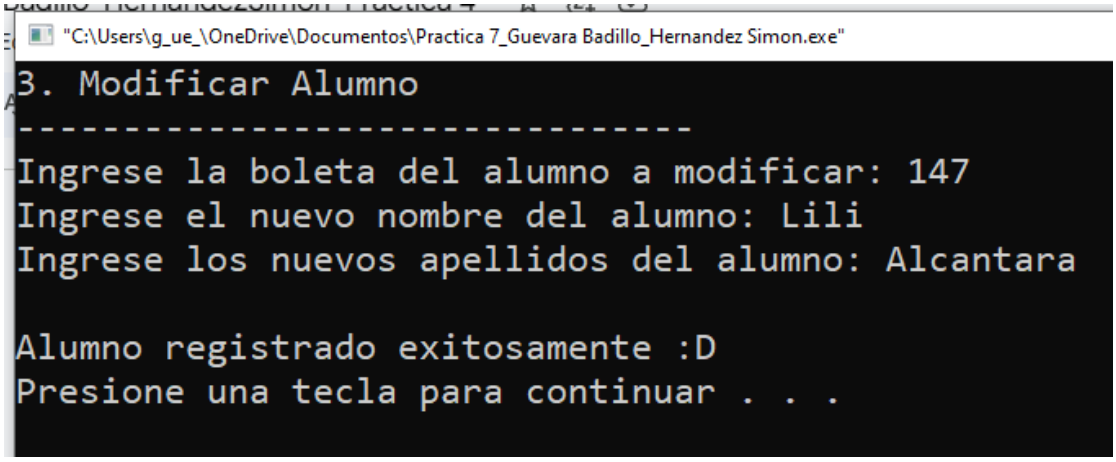
➔ Ascendente:

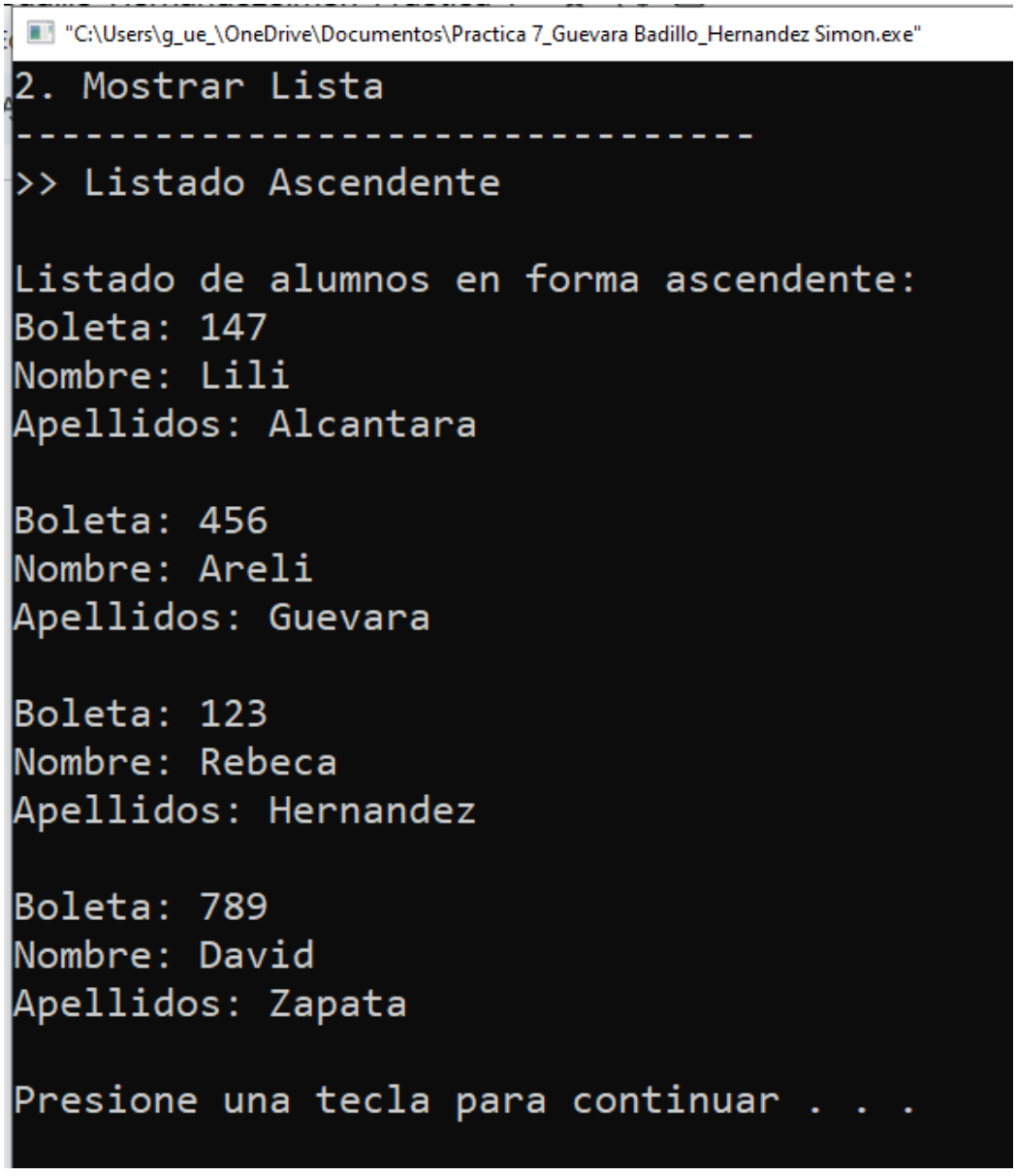


➔ Descendente:



Modificar:


"C:\Users\g_ue_\OneDrive\Documentos\Practica 7_Guevara Badillo_Hernandez Simon.exe"
3. Modificar Alumno
Ingrese la boleta del alumno a modificar: 147
Ingrese el nuevo nombre del alumno: Lili
Ingrese los nuevos apellidos del alumno: Alcantara
Alumno registrado exitosamente :D
Presione una tecla para continuar . . .


"C:\Users\g_ue_\OneDrive\Documentos\Practica 7_Guevara Badillo_Hernandez Simon.exe"
2. Mostrar Lista
>> Listado Ascendente
Listado de alumnos en forma ascendente:
Boleta: 147
Nombre: Lili
Apellidos: Alcantara
Boleta: 456
Nombre: Areli
Apellidos: Guevara
Boleta: 123
Nombre: Rebeca
Apellidos: Hernandez
Boleta: 789
Nombre: David
Apellidos: Zapata
Presione una tecla para continuar . . .