Listas ligadas dobles

Parcial	AutoEstudio

✅ ¿Qué son las listas doblemente ligadas?

Una **lista doblemente ligada** es una estructura de datos donde **cada nodo apunta al siguiente y al anterior**. Esto permite recorrer la lista **en ambos sentidos** (adelante y atrás).

📦 Estructura de un nodo:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Estructura de un nodo de lista doblemente ligada
typedef struct Nodo {
    int dato;               // Dato almacenado (puede ser struct en lugar de int)
    struct Nodo* siguiente;    // Apunta al siguiente nodo
    struct Nodo* anterior;     // Apunta al nodo anterior
} Nodo;
```

📂 Variables de inicio y final de la lista

```
Nodo* inicio = NULL;  // Apunta al primer nodo de la lista
Nodo* fin = NULL;     // Apunta al último nodo de la lista
```

🔷 Función 1: Inserción al inicio

```
bool insertarInicio(Nodo** inicio, Nodo** fin, int valor) {
    Nodo* nuevo = malloc(sizeof(Nodo));   // Reservamos memoria para el nuevo
```

```
    if (!nuevo) return false;           // Si falla la memoria, salimos

    nuevo->dato = valor;                // Guardamos el dato
    nuevo->anterior = NULL;             // No hay nodo antes, es el nuevo inicio
    nuevo->siguiente = *inicio;         // Su siguiente será el nodo actual de inicio

    if (*inicio != NULL)                // Si la lista no está vacía
        (*inicio)->anterior = nuevo;    // El nodo que era primero ahora apunta hacia

    else
        *fin = nuevo;                   // Si la lista estaba vacía, también es el final

    *inicio = nuevo;                    // Actualizamos el puntero de inicio
    return true;
}
```

📌 ¿Qué hace?

- Crea un nuevo nodo al **inicio**.
- Si la lista está vacía, es **inicio y fin**.
- Si no, lo conecta con el nodo anterior y actualiza punteros.

🔷 Función 2: Inserción al final

```
bool insertarFinal(Nodo** inicio, Nodo** fin, int valor) {
    Nodo* nuevo = malloc(sizeof(Nodo));   // Creamos el nuevo nodo
    if (!nuevo) return false;

    nuevo->dato = valor;
    nuevo->siguiente = NULL;            // No tiene siguiente (es el último)
    nuevo->anterior = *fin;             // Su anterior es el nodo que era último

    if (*fin != NULL)
        (*fin)->siguiente = nuevo;      // El nodo actual de fin apunta al nuevo
    else
```

```
        *inicio = nuevo;                // Si estaba vacía, también es el inicio

    *fin = nuevo;                       // Actualizamos el puntero fin
    return true;
}
```

📌 ¿Qué hace?

- Crea un nuevo nodo al **final**.
- Si la lista está vacía, también es el inicio.
- Conecta correctamente el nuevo nodo con el anterior.

🔷 Función 3: Inserción en una posición específica

```
bool insertarEnPosicion(Nodo** inicio, Nodo** fin, int valor, int pos) {
    if (pos < 0) return false;

    if (pos == 0)
        return insertarInicio(inicio, fin, valor);  // Insertar al inicio si posición es 0

    Nodo* actual = *inicio;
    int i = 0;

    while (actual != NULL && i < pos - 1) {         // Recorremos hasta llegar a la posi
        actual = actual->siguiente;
        i++;
    }

    if (actual == NULL)
        return insertarFinal(inicio, fin, valor);   // Si llegamos al final, insertamos al fi

    Nodo* nuevo = malloc(sizeof(Nodo));
    if (!nuevo) return false;

    nuevo->dato = valor;
```

```
    nuevo->siguiente = actual->siguiente;          // El nuevo apunta al siguiente del
    nuevo->anterior = actual;                      // El nuevo apunta atrás al actual

    if (actual->siguiente != NULL)
        actual->siguiente->anterior = nuevo;       // El nodo siguiente apunta atrás al
    else
        *fin = nuevo;                              // Si no hay siguiente, es el nuevo fin

    actual->siguiente = nuevo;                     // El nodo actual apunta al nuevo
    return true;
}
```

📌 ¿Qué hace?

- Inserta un nodo en una posición dada (`0 = inicio`).
- Recorre hasta la posición indicada.
- Ajusta punteros anteriores y siguientes correctamente.

🔷 Función 4: Eliminación al inicio

```
bool eliminarInicio(Nodo** inicio, Nodo** fin) {
    if (*inicio == NULL) return false;    // Lista vacía

    Nodo* temp = *inicio;
    *inicio = temp->siguiente;            // Saltamos al siguiente nodo

    if (*inicio != NULL)
        (*inicio)->anterior = NULL;       // El nuevo inicio ya no tiene anterior
    else
        *fin = NULL;                      // Si quedó vacía, fin también es NULL

    free(temp);                           // Liberamos memoria
    temp=NULL;
```

```
    return true;
}
```

📌 ¿Qué hace?

- Elimina el nodo del **inicio**.
- Si queda vacío, también actualiza el final.
- Libera memoria correctamente.

🔷 Función 5: Eliminación al final

```
bool eliminarFinal(Nodo** inicio, Nodo** fin) {
    if (*fin == NULL) return false;              // Lista vacía

    Nodo* temp = *fin;
    *fin = temp→anterior;                    // Retrocedemos al nodo anterior

    if (*fin != NULL)
        (*fin)→siguiente = NULL;              // El nuevo fin ya no tiene siguiente
    else
        *inicio = NULL;                      // Si quedó vacía

    free(temp);
    temp=NULL;
    return true;
}
```

📌 ¿Qué hace?

- Elimina el nodo del **final**.
- Actualiza el puntero fin.
- Si queda vacía, inicio también es NULL.

🔷 Función 6: Eliminación en una posición específica

```
bool eliminarEnPosicion(Nodo** inicio, Nodo** fin, int pos) {
    if (*inicio == NULL || pos < 0) return false;

    Nodo* actual = *inicio;
    int i = 0;

    while (actual != NULL && i < pos) {
        actual = actual->siguiente;
        i++;
    }

    if (actual == NULL) return false;

    if (actual->anterior != NULL)
        actual->anterior->siguiente = actual->siguiente;
    else
        *inicio = actual->siguiente;              // Si eliminamos el primero

    if (actual->siguiente != NULL)
        actual->siguiente->anterior = actual->anterior;
    else
        *fin = actual->anterior;                  // Si eliminamos el último

    free(actual);
    actual = NULL;
    return true;
}
```

📌 ¿Qué hace?

- Elimina el nodo en la **posición** `pos`.
- Ajusta enlaces del anterior y siguiente correctamente.
- Actualiza `inicio` y `fin` si es necesario.

🔷 Función 7: Recorrido hacia adelante

```
void recorrerLista(Nodo* inicio) {
    Nodo* actual = inicio;
    while (actual != NULL) {
        printf("%d ", actual->dato);
        actual = actual->siguiente;
    }
    printf("\n");
}
```

📌 ¿Qué hace?

- Recorre e imprime los datos desde el inicio hasta el final.

🔷 Función 8: Recorrido hacia atrás

```
void recorrerReversa(Nodo* fin) {
    Nodo* actual = fin;
    while (actual != NULL) {
        printf("%d ", actual->dato);
        actual = actual->anterior;
    }
    printf("\n");
}
```

📌 ¿Qué hace?

- Recorre e imprime desde el final hasta el inicio.

Ejemplo de prueba usando int

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Definición de nodo
typedef struct Nodo {
    int dato;
    struct Nodo* siguiente;
    struct Nodo* anterior;
} Nodo;

// Variables globales para inicio y fin
Nodo* inicio = NULL;
Nodo* fin = NULL;

// Funciones
bool insertarInicio(int valor) {
    Nodo* nuevo = malloc(sizeof(Nodo));
    if (!nuevo) return false;

    nuevo->dato = valor;
    nuevo->anterior = NULL;
    nuevo->siguiente = inicio;

    if (inicio != NULL)
        inicio->anterior = nuevo;
    else
        fin = nuevo;

    inicio = nuevo;
    return true;
}

bool insertarFinal(int valor) {
    Nodo* nuevo = malloc(sizeof(Nodo));
```

```
    if (!nuevo) return false;

    nuevo->dato = valor;
    nuevo->siguiente = NULL;
    nuevo->anterior = fin;

    if (fin != NULL)
        fin->siguiente = nuevo;
    else
        inicio = nuevo;

    fin = nuevo;
    return true;
}

bool insertarEnPosicion(int valor, int pos) {
    if (pos < 0) return false;
    if (pos == 0) return insertarInicio(valor);

    Nodo* actual = inicio;
    int i = 0;
    while (actual != NULL && i < pos - 1) {
        actual = actual->siguiente;
        i++;
    }

    if (actual == NULL) return insertarFinal(valor);

    Nodo* nuevo = malloc(sizeof(Nodo));
    if (!nuevo) return false;

    nuevo->dato = valor;
    nuevo->siguiente = actual->siguiente;
    nuevo->anterior = actual;

    if (actual->siguiente != NULL)
```

```
        actual->siguiente->anterior = nuevo;
    else
        fin = nuevo;

    actual->siguiente = nuevo;
    return true;
}

bool eliminarInicio() {
    if (inicio == NULL) return false;

    Nodo* temp = inicio;
    inicio = inicio->siguiente;

    if (inicio != NULL)
        inicio->anterior = NULL;
    else
        fin = NULL;

    free(temp);
    temp = NULL;
    return true;
}

bool eliminarFinal() {
    if (fin == NULL) return false;

    Nodo* temp = fin;
    fin = fin->anterior;

    if (fin != NULL)
        fin->siguiente = NULL;
    else
        inicio = NULL;

    free(temp);
```

```
    temp = NULL;
    return true;
}

bool eliminarEnPosicion(int pos) {
    if (inicio == NULL || pos < 0) return false;

    Nodo* actual = inicio;
    int i = 0;
    while (actual != NULL && i < pos) {
        actual = actual->siguiente;
        i++;
    }

    if (actual == NULL) return false;

    if (actual->anterior != NULL)
        actual->anterior->siguiente = actual->siguiente;
    else
        inicio = actual->siguiente;

    if (actual->siguiente != NULL)
        actual->siguiente->anterior = actual->anterior;
    else
        fin = actual->anterior;

    free(actual);
    actual = NULL;
    return true;
}

void recorrerLista() {
    Nodo* actual = inicio;
    printf("Lista: ");
    while (actual != NULL) {
        printf("%d ", actual->dato);
```

```
        actual = actual->siguiente;
    }
    printf("\n");
}

void recorrerReversa() {
    Nodo* actual = fin;
    printf("Lista (reversa): ");
    while (actual != NULL) {
        printf("%d ", actual->dato);
        actual = actual->anterior;
    }
    printf("\n");
}

// Función principal con menú
int main() {
    int opcion, valor, posicion;

    do {
        printf("\n--- MENU LISTA DOBLE ---\n");
        printf("1. Insertar al inicio\n");
        printf("2. Insertar al final\n");
        printf("3. Insertar en posicion\n");
        printf("4. Eliminar al inicio\n");
        printf("5. Eliminar al final\n");
        printf("6. Eliminar en posicion\n");
        printf("7. Mostrar lista\n");
        printf("8. Mostrar reversa\n");
        printf("0. Salir\n");
        printf("Selecciona una opcion: ");
        scanf("%d", &opcion);

        switch (opcion) {
            case 1:
                printf("Valor a insertar: ");
```

```
            scanf("%d", &valor);
            insertarInicio(valor);
            break;
        case 2:
            printf("Valor a insertar: ");
            scanf("%d", &valor);
            insertarFinal(valor);
            break;
        case 3:
            printf("Valor a insertar: ");
            scanf("%d", &valor);
            printf("Posicion: ");
            scanf("%d", &posicion);
            insertarEnPosicion(valor, posicion);
            break;
        case 4:
            eliminarInicio();
            break;
        case 5:
            eliminarFinal();
            break;
        case 6:
            printf("Posicion a eliminar: ");
            scanf("%d", &posicion);
            eliminarEnPosicion(posicion);
            break;
        case 7:
            recorrerLista();
            break;
        case 8:
            recorrerReversa();
            break;
        case 0:
            printf("Saliendo...\n");
            break;
        default:
```

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

    return 0;
}
```

Aplicado a estudiantes

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

// Estructura Estudiante
typedef struct {
    int matricula;
    char nombre[30];
    bool regular;
} Estudiante;

// Nodo doblemente ligado con Estudiante
typedef struct Nodo {
    Estudiante dato;
    struct Nodo* siguiente;
    struct Nodo* anterior;
} Nodo;

// Punteros globales
Nodo* inicio = NULL;
Nodo* fin = NULL;

// Función para capturar un estudiante desde teclado
```

```
Estudiante capturarEstudiante() {
    Estudiante e;
    printf("Matricula: ");
    scanf("%d", &e.matricula);
    printf("Nombre: ");
    getchar(); // limpiar buffer
    fgets(e.nombre, sizeof(e.nombre), stdin);
    e.nombre[strcspn(e.nombre, "\n")] = '\0'; // quitar salto de línea
    printf("¿Es regular? (1=Sí, 0=No): ");
    scanf("%d", (int*)&e.regular);
    return e;
}

// Función para imprimir un estudiante
void imprimirEstudiante(Estudiante e) {
    printf("[Matricula: %d, Nombre: %s, Regular: %s] ",
        e.matricula, e.nombre, e.regular ? "Sí" : "No");
}

// Inserción al inicio
bool insertarInicio(Estudiante valor) {
    Nodo* nuevo = malloc(sizeof(Nodo));
    if (!nuevo) return false;
    nuevo->dato = valor;
    nuevo->anterior = NULL;
    nuevo->siguiente = inicio;

    if (inicio != NULL)
        inicio->anterior = nuevo;
    else
        fin = nuevo;

    inicio = nuevo;
    return true;
}
```

```
// Inserción al final
bool insertarFinal(Estudiante valor) {
    Nodo* nuevo = malloc(sizeof(Nodo));
    if (!nuevo) return false;
    nuevo→dato = valor;
    nuevo→siguiente = NULL;
    nuevo→anterior = fin;

    if (fin != NULL)
        fin→siguiente = nuevo;
    else
        inicio = nuevo;

    fin = nuevo;
    return true;
}

// Inserción en posición
bool insertarEnPosicion(Estudiante valor, int pos) {
    if (pos < 0) return false;
    if (pos == 0) return insertarInicio(valor);

    Nodo* actual = inicio;
    int i = 0;
    while (actual != NULL && i < pos - 1) {
        actual = actual→siguiente;
        i++;
    }

    if (actual == NULL) return insertarFinal(valor);

    Nodo* nuevo = malloc(sizeof(Nodo));
    if (!nuevo) return false;

    nuevo→dato = valor;
    nuevo→siguiente = actual→siguiente;
```

```
    nuevo->anterior = actual;

    if (actual->siguiente != NULL)
        actual->siguiente->anterior = nuevo;
    else
        fin = nuevo;

    actual->siguiente = nuevo;
    return true;
}

// Eliminación al inicio
bool eliminarInicio() {
    if (inicio == NULL) return false;

    Nodo* temp = inicio;
    inicio = inicio->siguiente;

    if (inicio != NULL)
        inicio->anterior = NULL;
    else
        fin = NULL;

    free(temp);
    temp = NULL;
    return true;
}

// Eliminación al final
bool eliminarFinal() {
    if (fin == NULL) return false;

    Nodo* temp = fin;
    fin = fin->anterior;

    if (fin != NULL)
```

```
        fin->siguiente = NULL;
    else
        inicio = NULL;

    free(temp);
    temp = NULL;
    return true;
}

// Eliminación en posición
bool eliminarEnPosicion(int pos) {
    if (inicio == NULL || pos < 0) return false;

    Nodo* actual = inicio;
    int i = 0;
    while (actual != NULL && i < pos) {
        actual = actual->siguiente;
        i++;
    }

    if (actual == NULL) return false;

    if (actual->anterior != NULL)
        actual->anterior->siguiente = actual->siguiente;
    else
        inicio = actual->siguiente;

    if (actual->siguiente != NULL)
        actual->siguiente->anterior = actual->anterior;
    else
        fin = actual->anterior;

    free(actual);
    actual = NULL;
    return true;
}
```

```
// Recorrido hacia adelante
void recorrerLista() {
    Nodo* actual = inicio;
    printf("\nLista de estudiantes:\n");
    while (actual != NULL) {
        imprimirEstudiante(actual->dato);
        printf("\n");
        actual = actual->siguiente;
    }
}

// Recorrido hacia atrás
void recorrerReversa() {
    Nodo* actual = fin;
    printf("\nLista en reversa:\n");
    while (actual != NULL) {
        imprimirEstudiante(actual->dato);
        printf("\n");
        actual = actual->anterior;
    }
}

// Menú principal
int main() {
    int opcion, posicion;
    Estudiante e;

    do {
        printf("\n--- MENU LISTA DOBLE DE ESTUDIANTES ---\n");
        printf("1. Insertar al inicio\n");
        printf("2. Insertar al final\n");
        printf("3. Insertar en posicion\n");
        printf("4. Eliminar al inicio\n");
        printf("5. Eliminar al final\n");
        printf("6. Eliminar en posicion\n");
```

```
        printf("7. Mostrar lista\n");
        printf("8. Mostrar reversa\n");
        printf("0. Salir\n");
        printf("Selecciona una opcion: ");
        scanf("%d", &opcion);

        switch (opcion) {
            case 1:
                e = capturarEstudiante();
                insertarInicio(e);
                break;
            case 2:
                e = capturarEstudiante();
                insertarFinal(e);
                break;
            case 3:
                e = capturarEstudiante();
                printf("Posicion: ");
                scanf("%d", &posicion);
                insertarEnPosicion(e, posicion);
                break;
            case 4:
                eliminarInicio();
                break;
            case 5:
                eliminarFinal();
                break;
            case 6:
                printf("Posicion a eliminar: ");
                scanf("%d", &posicion);
                eliminarEnPosicion(posicion);
                break;
            case 7:
                recorrerLista();
                break;
            case 8:
```

```
            recorrerReversa();
            break;
        case 0:
            printf("Saliendo...\n");
            break;
        default:
            printf("Opcion invalida.\n");
      }
   } while (opcion != 0);

   return 0;
}
```