

Listas ligadas simples

⌵ Parcial

AutoEstudio

Una **lista ligada** (o enlazada) es una estructura de datos lineal donde cada elemento (llamado **nodo**) contiene dos partes:

- **Dato** (por ejemplo, un número o una estructura).
- **Apuntador** (puntero) al siguiente nodo de la lista.

A diferencia de los arreglos, las listas ligadas **no tienen un tamaño fijo** y los elementos **no están en posiciones contiguas en memoria**.



Analogía

Imagina una lista ligada como un tren:

- Cada vagón tiene un número (dato) y un enganche al siguiente vagón (puntero).
- No necesitas saber cuántos vagones hay.
- Puedes agregar o quitar vagones en cualquier parte del tren.



Definición de un Nodo en C

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

typedef struct Nodo {
    int dato;
    struct Nodo* siguiente;
} Nodo;
```

El `typedef` permite luego declarar nodos como `Nodo*`.



1. Inserción al inicio de una lista

```
bool insertarInicio(Nodo** cabeza, int valor) {
    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    if (nuevo == NULL) return false;
    nuevo->dato = valor;
    nuevo->siguiente = *cabeza;
    *cabeza = nuevo;
    return true;
}
```

¿Qué hace?

Agrega un nuevo nodo al **inicio de la lista**.

Pasos:

1. Reserva memoria para un nuevo nodo.
2. Si `malloc` falla, retorna `false`.
3. Coloca el valor en el nuevo nodo.
4. Apunta el nuevo nodo al nodo que antes era el primero.
5. Actualiza la cabeza de la lista para que apunte al nuevo nodo.
6. Retorna `true`.



Piensa en insertar un libro al principio de una pila.



Ejemplo visual:

```
ANTES:    cabeza → [3] → [5] → NULL
NUEVO VALOR: 1
DESPUÉS:  cabeza → [1] → [3] → [5] → NULL
```



2. Inserción al final de una lista

```

bool insertarFinal(Nodo** cabeza, int valor) {
    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    if (nuevo == NULL) return false;
    nuevo→dato = valor;
    nuevo→siguiente = NULL;

    if (*cabeza == NULL) {
        *cabeza = nuevo;
        return true;
    }

    Nodo* actual = *cabeza;
    while (actual→siguiente != NULL) {
        actual = actual→siguiente;
    }
    actual→siguiente = nuevo;
    return true;
}

```

¿Qué hace?

Agrega un nodo al **final de la lista**.

Pasos:

1. Crea un nuevo nodo con `valor`.
2. Si la lista está vacía, lo pone como primer nodo.
3. Si no, recorre la lista hasta llegar al último nodo.
4. El último nodo ahora apunta al nuevo nodo.
5. Retorna `true`.

 Ejemplo:

ANTES: cabeza → [3] → [5] → NULL
 NUEVO VALOR: 8

DESPUÉS: cabeza → [3] → [5] → [8] → NULL

3. Inserción en una posición específica

```
bool insertarEnPosicion(Nodo** cabeza, int valor, int posicion) {
    if (posicion < 0) return false;
    if (posicion == 0) return insertarInicio(cabeza, valor);

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

    while (actual != NULL && i < posicion - 1) {
        actual = actual→siguiente;
        i++;
    }

    if (actual == NULL) return false;

    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    if (nuevo == NULL) return false;
    nuevo→dato = valor;
    nuevo→siguiente = actual→siguiente;
    actual→siguiente = nuevo;
    return true;
}
```

¿Qué hace?

Agrega un nodo en una **posición específica**.

Pasos:

1. Si `posición` es 0, llama a `insertarInicio`.
2. Recorre la lista hasta el nodo anterior a la posición deseada.
3. Si no encuentra la posición válida, retorna `false`.

4. Inserta el nuevo nodo en medio de la lista.


5. Retorna `true`.

 Ejemplo:

ANTES: cabeza → [3] → [5] → NULL

INSERTA: 4 en posición 1

DESPUÉS: cabeza → [3] → [4] → [5] → NULL

 Si la posición no existe, muestra un mensaje de error.

4. Eliminación al inicio

```
bool eliminarInicio(Nodo** cabeza) {  
    if (*cabeza == NULL) return false;  
    Nodo* temp = *cabeza;  
    *cabeza = temp->siguiente;  
    free(temp);  
    temp = NULL;  
    return true;  
}
```

¿Qué hace?

Elimina el **primer nodo** de la lista.

Pasos:

1. Si la lista está vacía, retorna `false`.
2. Guarda una referencia temporal del primer nodo.
3. Actualiza la cabeza al siguiente nodo.
4. Libera la memoria del nodo eliminado y lo manda a `NULL`.
5. Retorna `true`.

 Ejemplo:

ANTES: cabeza → [3] → [5] → NULL
DESPUÉS: cabeza → [5] → NULL

! 5. Eliminación al final

```
bool eliminarFinal(Nodo** cabeza) {  
    if (*cabeza == NULL) return false;  
  
    if ((*cabeza)→siguiente == NULL) {  
        free(*cabeza);  
        *cabeza = NULL;  
        return true;  
    }  
  
    Nodo* actual = *cabeza;  
    while (actual→siguiente→siguiente != NULL) {  
        actual = actual→siguiente;  
    }  
  
    free(actual→siguiente);  
    actual→siguiente = NULL;  
    return true;  
}
```

¿Qué hace?

Elimina el **último nodo**.

Pasos:

1. Si la lista está vacía, retorna `false`.
2. Si solo hay un nodo, lo elimina y pone `NULL` como cabeza.
3. Si hay más de uno, recorre hasta el penúltimo.
4. Libera el último nodo.

5. Retorna `true`.

 Ejemplo:

ANTES: cabeza → [3] → [5] → [8] → NULL

DESPUÉS: cabeza → [3] → [5] → NULL

6. Eliminación en una posición específica

```
bool eliminarEnPosicion(Nodo** cabeza, int posicion) {
    if (*cabeza == NULL || posicion < 0) return false;
    if (posicion == 0) return eliminarInicio(cabeza);

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

    while (actual->siguiente != NULL && i < posicion - 1) {
        actual = actual->siguiente;
        i++;
    }

    if (actual->siguiente == NULL) return false;

    Nodo* temp = actual->siguiente;
    actual->siguiente = temp->siguiente;
    free(temp);
    temp = NULL;
    return true;
}
```

¿Qué hace?

Elimina un nodo en una **posición específica**.

Pasos:

1. Si la lista está vacía o la posición es inválida, retorna `false`.
2. Si es la posición 0, llama a `eliminarInicio`.
3. Recorre hasta el nodo anterior a la posición.
4. Elimina el nodo apuntado, ajusta los enlaces.
5. Libera la memoria del nodo eliminado.
6. Retorna `true`.

 Ejemplo:

ANTES: cabeza → [3] → [5] → [8] → NULL
ELIMINA en posición 1
DESPUÉS: cabeza → [3] → [8] → NULL

7. Recorrer la lista


```
void imprimirLista(Nodo* cabeza) {  
    Nodo* actual = cabeza;  
    while (actual != NULL) {  
        printf("%d → ", actual→dato);  
        actual = actual→siguiente;  
    }  
    printf("NULL\n");  
}
```

¿Qué hace?

Imprime todos los nodos de la lista en orden.

Pasos:

1. Recorre desde el primer nodo hasta el último.
2. Imprime cada dato seguido de `>`.
3. Al final imprime `NULL`.

 Ejemplo de salida:

3 → 5 → 8 → NULL

Ejemplo de uso

```
int main() {
    Nodo* lista = NULL;

    insertarInicio(&lista, 10);
    insertarFinal(&lista, 20);
    insertarEnPosicion(&lista, 15, 1); // Inserta 15 en la posición 1

    recorrerLista(lista); // 10 → 15 → 20 → NULL

    eliminarEnPosicion(&lista, 1); // Elimina el nodo en la posición 1
    recorrerLista(lista); // 10 → 20 → NULL

    eliminarInicio(&lista);
    eliminarFinal(&lista);
    recorrerLista(lista); // NULL

    return 0;
}
```

aplicandolo para trabajar con estructura de estudiantes

listas.h

```
#ifndef LISTA_H
#define LISTA_H
```

```

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

// Tipo genérico de nodo usando un tipo definido externamente llamado TIPO_DATO
#define TIPO_DATO Estudiante

typedef struct Nodo {
    TIPO_DATO dato;
    struct Nodo* siguiente;
} Nodo;

// Funciones genéricas
bool insertarInicio(Nodo** cabeza, TIPO_DATO valor) {
    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    if (nuevo == NULL) return false;
    nuevo->dato = valor;
    nuevo->siguiente = *cabeza;
    *cabeza = nuevo;
    return true;
}

bool insertarFinal(Nodo** cabeza, TIPO_DATO valor) {
    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    if (nuevo == NULL) return false;
    nuevo->dato = valor;
    nuevo->siguiente = NULL;

    if (*cabeza == NULL) {
        *cabeza = nuevo;
        return true;
    }

    Nodo* actual = *cabeza;
    while (actual->siguiente != NULL) {

```

```

        actual = actual→siguiente;
    }
    actual→siguiente = nuevo;
    return true;
}

bool insertarEnPosicion(Nodo** cabeza, TIPO_DATO valor, int posicion) {
    if (posicion < 0) return false;
    if (posicion == 0) return insertarInicio(cabeza, valor);

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

    while (actual != NULL && i < posicion - 1) {
        actual = actual→siguiente;
        i++;
    }

    if (actual == NULL) return false;

    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    if (nuevo == NULL) return false;
    nuevo→dato = valor;
    nuevo→siguiente = actual→siguiente;
    actual→siguiente = nuevo;
    return true;
}

bool eliminarInicio(Nodo** cabeza) {
    if (*cabeza == NULL) return false;
    Nodo* temp = *cabeza;
    *cabeza = temp→siguiente;
    free(temp);
    return true;
}

```

```

bool eliminarFinal(Nodo** cabeza) {
    if (*cabeza == NULL) return false;

    if ((*cabeza)→siguiente == NULL) {
        free(*cabeza);
        *cabeza = NULL;
        return true;
    }

    Nodo* actual = *cabeza;
    while (actual→siguiente→siguiente != NULL) {
        actual = actual→siguiente;
    }

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

bool eliminarEnPosicion(Nodo** cabeza, int posicion) {
    if (*cabeza == NULL || posicion < 0) return false;
    if (posicion == 0) return eliminarInicio(cabeza);

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

    while (actual→siguiente != NULL && i < posicion - 1) {
        actual = actual→siguiente;
        i++;
    }

    if (actual→siguiente == NULL) return false;

    Nodo* temp = actual→siguiente;
    actual→siguiente = temp→siguiente;
    free(temp);
}

```

```

    return true;
}

// El recorrido lo dejamos para que el usuario lo defina según su estructura

#endif

```

listas.c

```

#include <stdio.h>
#include <string.h>
#include "lista.h"

// Definimos el tipo de dato antes de incluir la cabecera
typedef struct {
    int matricula;
    char nombre[50];
    char apellido[50];
    int edad;
    bool esRegular;
} Estudiante;

#undef TIPO_DATO
#define TIPO_DATO Estudiante
#include "lista.h"

// Función para imprimir un estudiante
void imprimirEstudiante(Estudiante e) {
    printf("Matricula: %d\n", e.matricula);
    printf("Nombre: %s %s\n", e.nombre, e.apellido);
    printf("Edad: %d\n", e.edad);
    printf("Regular: %s\n", e.esRegular ? "Sí" : "No");
}

// Función para recorrer lista

```

```

void recorrerLista(Nodo* cabeza) {
    Nodo* actual = cabeza;
    while (actual != NULL) {
        imprimirEstudiante(actual→dato);
        printf("-----\n");
        actual = actual→siguiente;
    }
}

```

```

Estudiante crearEstudianteDesdeTeclado() {
    Estudiante e;
    printf("Matricula: "); scanf("%d", &e.matricula);
    printf("Nombre: "); scanf(" %s", e.nombre);
    printf("Apellido: "); scanf(" %s", e.apellido);
    printf("Edad: "); scanf("%d", &e.edad);
    int regular;
    printf("¿Es regular? (1: Sí, 0: No): "); scanf("%d", &regular);
    e.esRegular = (regular == 1);
    return e;
}

```

```

int main() {
    Nodo* lista = NULL;
    int opcion;

    do {
        printf("\n--- Menú ---\n");
        printf("1. Insertar estudiante al inicio\n");
        printf("2. Insertar estudiante al final\n");
        printf("3. Insertar estudiante en posición\n");
        printf("4. Eliminar estudiante al inicio\n");
        printf("5. Eliminar estudiante al final\n");
        printf("6. Eliminar estudiante en posición\n");
        printf("7. Ver estudiantes\n");
        printf("0. Salir\n");
        printf("Opción: ");
    } while (opcion != 0);
}

```

```

scanf("%d", &opcion);

Estudiante e;
int pos;

switch (opcion) {
    case 1:
        e = crearEstudianteDesdeTeclado();
        insertarInicio(&lista, e);
        break;
    case 2:
        e = crearEstudianteDesdeTeclado();
        insertarFinal(&lista, e);
        break;
    case 3:
        e = crearEstudianteDesdeTeclado();
        printf("Posición: ");
        scanf("%d", &pos);
        insertarEnPosicion(&lista, e, pos);
        break;
    case 4:
        eliminarInicio(&lista);
        break;
    case 5:
        eliminarFinal(&lista);
        break;
    case 6:
        printf("Posición: ");
        scanf("%d", &pos);
        eliminarEnPosicion(&lista, pos);
        break;
    case 7:
        recorrerLista(lista);
        break;
    case 0:
        printf("Saliendo...\n");

```

```
        break;
    default:
        printf("Opción inválida.\n");
    }
} while (opcion != 0);

return 0;
}
```