



# UNIVERSIDAD POLITÉCNICA DE VICTORIA

**06/11/25**

**Nombre de la Materia:**

Estructura de Datos

## Actividad 2 - Listas doblemente enlazadas

**Carrera:**

Ingeniería en Tecnologías de la Información e  
Innovación Digital.

**Alumno:**

Eliezer Mores Oyervides  
2430037

**Catedrático:**

**DR. SAID POLANCO MARTAGÓN**

septiembre - diciembre de 2025



## Pantallazos de implementaciones del código:

```
class TramaLoad : public TramaBase {
private:
    char caracter; // Carácter a decodificar

public:
    /**
     * @brief Constructor
     * @param c Carácter de la trama
     */
    TramaLoad(char c) : caracter(c) {}

    /**
     * @brief Obtiene el carácter de la trama
     * @return Carácter almacenado
     */
    char getCaracter() const { return caracter; }
};

/** 
 * @class TramaMap
 * @brief Representa una trama de mapeo (M,N)
 *
 * Contiene una instrucción para rotar el disco de cifrado,
 * cambiando el mapeo para las futuras tramas LOAD
 */
class TramaMap : public TramaBase {
private:
    int rotacion; // Número de posiciones a rotar

public:
    /**
     * @brief Constructor
     * @param n Número de posiciones a rotar
     */
    TramaMap(int n) : rotacion(n) {}

    /**
     * @brief Obtiene el valor de rotación
     * @return Número de posiciones a rotar
     */
    int getRotacion() const { return rotacion; }
};

#endif // TRAMAS_H
```

```
bool SerialReader::leerLinea(char* buffer, int maxLen) {
    if (!conectado || puerto < 0) return false;

    int pos = 0;

    while (pos < maxLen - 1) {
        char c;
        int n = read(puerto, &c, 1);

        if (n < 0) {
            // Error de lectura
            std::cerr << "Error al leer del puerto serial" << std::endl;
            return false;
        } else if (n == 0) {
            // No hay datos disponibles, continuar esperando
            continue;
        }

        // Verificar fin de línea
        if (c == '\n' || c == '\r') {
            if (pos > 0) {
                buffer[pos] = '\0';
                return true;
            }
            // Ignorar líneas vacías
        } else {
            buffer[pos++] = c;
        }
    }

    // Buffer lleno
    buffer[pos] = '\0';
    return pos > 0;
}

void SerialReader::cerrar() {
    if (conectado && puerto >= 0) {
        close(puerto);
        puerto = -1;
        conectado = false;
    }
}
```



```
RotorDeMapeo::~RotorDeMapeo() {
    if (cabeza == nullptr) return;

    // Romper el círculo
    NodoRotor* ultimo = cabeza->previo;
    ultimo->siguiente = nullptr;

    // Eliminar todos los nodos
    NodoRotor* actual = cabeza;
    while (actual != nullptr) {
        NodoRotor* siguiente = actual->siguiente;
        delete actual;
        actual = siguiente;
    }
}

void RotorDeMapeo::rotar(int n) {
    if (cabeza == nullptr) return;

    // Normalizar n al rango del tamaño
    n = n % tamaño;
    if (n < 0) n += tamaño;

    // Rotar moviendo la cabeza
    for (int i = 0; i < n; i++) {
        cabeza = cabeza->siguiente;
    }
}

char RotorDeMapeo::getMapeo(char in) {
    if (cabeza == nullptr) return in;

    // Convertir a mayúscula si es letra
    if (in >= 'a' && in <= 'z') {
        in = in - 'a' + 'A';
    }

    // IMPORTANTE: El espacio NO se cifra, siempre retorna espacio
    if (in == ' ') {
        return ' ';
    }

    // Si no es una letra mayúscula, devolver sin cambios
    if (in < 'A' || in > 'Z') {
        return in;
    }

    // Crear un rotor de referencia (sin rotación) para encontrar posición original
    const char alfabeto[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

```
ListaDeCarga::ListaDeCarga() : cabeza(nullptr), cola(nullptr), tamaño(0) {}

ListaDeCarga::~ListaDeCarga() {
    NodoCarga* actual = cabeza;
    while (actual != nullptr) {
        NodoCarga* siguiente = actual->siguiente;
        // Liberar la trama apuntada
        delete actual->trama;
        // Liberar el nodo
        delete actual;
        actual = siguiente;
    }
}

void ListaDeCarga::insertarAlFinal(TramaBase* trama) {
    NodoCarga* nuevo = new NodoCarga(trama);

    if (cabeza == nullptr) {
        // Lista vacía
        cabeza = nuevo;
        cola = nuevo;
    } else {
        // Insertar al final
        cola->siguiente = nuevo;
        nuevo->previo = cola;
        cola = nuevo;
    }

    tamaño++;
}

void ListaDeCarga::procesarTramas(RotorDeMapeo* rotor) {
    // Este método es para procesamiento batch (no se usa en el modo tiempo real)
    // pero se mantiene por si se necesita reprocesar la lista

    if (cabeza == nullptr) {
        std::cout << "[Lista vacía - sin tramas para procesar]" << std::endl;
        return;
    }

    // Buffer temporal para almacenar caracteres decodificados
    char* mensajeTemp = new char[tamaño + 1];
    int posiciónMensaje = 0;

    std::cout << "Procesando " << tamaño << " tramas almacenadas..." << std::endl;
    std::cout << "======" << std::endl;

    NodoCarga* actual = cabeza;
    int númeroTrama = 1;
```



## Ejecución del programa

```
Iniciando Decodificador PRT-7. Conectando a puerto...
Ingrese el nombre del puerto (ej: /dev/ttyUSB0): /dev/ttyUSB0
Conexión establecida. Esperando tramas...

Trama recibida: [L,H] -> Procesando... -> Fragmento 'H' decodificado como 'H'. Mensaje: [[H]]
Trama recibida: [L,O] -> Procesando... -> Fragmento 'O' decodificado como 'O'. Mensaje: [[H][O]]
Trama recibida: [L,L] -> Procesando... -> Fragmento 'L' decodificado como 'L'. Mensaje: [[H][O][L]]
Trama recibida: [M,2] -> Procesando... -> ROTANDO ROTOR +2. (Ahora 'A' se mapea a 'C')
Trama recibida: [L,A] -> Procesando... -> Fragmento 'A' decodificado como 'C'. Mensaje: [[H][O][L][C]]
Trama recibida: [L,Space] -> Procesando... -> Fragmento ' ' decodificado como ' '. Mensaje: [[H][O][L][C][ ]]
Trama recibida: [L,W] -> Procesando... -> Fragmento 'W' decodificado como 'Y'. Mensaje: [[H][O][L][C][ ][Y]]
Trama recibida: [M,-2] -> Procesando... -> ROTANDO ROTOR -2. (Ahora 'A' se mapea a 'A')
Trama recibida: [L,O] -> Procesando... -> Fragmento 'O' decodificado como 'O'. Mensaje: [[H][O][L][C][ ][Y][O]]
Trama recibida: [L,R] -> Procesando... -> Fragmento 'R' decodificado como 'R'. Mensaje: [[H][O][L][C][ ][Y][O][R]]
Trama recibida: [L,L] -> Procesando... -> Fragmento 'L' decodificado como 'L'. Mensaje: [[H][O][L][C][ ][Y][O][R][L]]
Trama recibida: [L,D] -> Procesando... -> Fragmento 'D' decodificado como 'D'. Mensaje: [[H][O][L][C][ ][Y][O][R][L][D]]

...
Flujo de datos terminado.
MENSAJE OCULTO ENSAMBLADO:
HOLC YORLD
...
Liberando memoria... Sistema apagado.
```

### Selección de puerto:

```
Iniciando Decodificador PRT-7. Conectando a puerto...
Ingrese el nombre del puerto (ej: /dev/ttyUSB0): /dev/ttyUSB0
Conexión establecida. Esperando tramas...
```

### Ingreso de datos:

```
Conexión establecida. Esperando tramas...

Trama recibida: [L,H] -> Procesando... -> Fragmento 'H' decodificado como 'H'. Mensaje: [[H]]
Trama recibida: [L,O] -> Procesando... -> Fragmento 'O' decodificado como 'O'. Mensaje: [[H][O]]
Trama recibida: [L,L] -> Procesando... -> Fragmento 'L' decodificado como 'L'. Mensaje: [[H][O][L]]
Trama recibida: [M,2] -> Procesando... -> ROTANDO ROTOR +2. (Ahora 'A' se mapea a 'C')
Trama recibida: [L,A] -> Procesando... -> Fragmento 'A' decodificado como 'C'. Mensaje: [[H][O][L][C]]
Trama recibida: [L,Space] -> Procesando... -> Fragmento ' ' decodificado como ' '. Mensaje: [[H][O][L][C][ ]]
Trama recibida: [L,W] -> Procesando... -> Fragmento 'W' decodificado como 'Y'. Mensaje: [[H][O][L][C][ ][Y]]
Trama recibida: [M,-2] -> Procesando... -> ROTANDO ROTOR -2. (Ahora 'A' se mapea a 'A')
Trama recibida: [L,O] -> Procesando... -> Fragmento 'O' decodificado como 'O'. Mensaje: [[H][O][L][C][ ][Y][O]]
Trama recibida: [L,R] -> Procesando... -> Fragmento 'R' decodificado como 'R'. Mensaje: [[H][O][L][C][ ][Y][O][R]]
Trama recibida: [L,L] -> Procesando... -> Fragmento 'L' decodificado como 'L'. Mensaje: [[H][O][L][C][ ][Y][O][R][L]]
Trama recibida: [L,D] -> Procesando... -> Fragmento 'D' decodificado como 'D'. Mensaje: [[H][O][L][C][ ][Y][O][R][L][D]]
```

### Rotación:

```
Trama recibida: [L,L] -> Procesando... -> Fragmento 'L' decodificado como 'L'. Mensaje: [[H][O][L]]
Trama recibida: [M,2] -> Procesando... -> ROTANDO ROTOR +2. (Ahora 'A' se mapea a 'C')
Trama recibida: [L,A] -> Procesando... -> Fragmento 'A' decodificado como 'C'. Mensaje: [[H][O][L][C]]
Trama recibida: [L,Space] -> Procesando... -> Fragmento ' ' decodificado como ' '. Mensaje: [[H][O][L][C][ ]]
```



### Regreso:

```
Trama recibida: [L,W] -> Procesando... -> Fragmento 'W' decodificado como 'Y'. Mensaje: [[H][0][L][C][ ][Y]]  
Trama recibida: [M,-2] -> Procesando... -> ROTANDO ROTOR -2. (Ahora 'A' se mapea a 'A')  
Trama recibida: [L,0] -> Procesando... -> Fragmento '0' decodificado como '0'. Mensaje: [[H][0][L][C][ ][Y][0]]
```

### Frase completa:

```
---  
Flujo de datos terminado.  
MENSAJE OCULTO ENSAMBLADO:  
HOLC YORLD  
---
```

### Fin del programa:

```
---  
Flujo de datos terminado.  
MENSAJE OCULTO ENSAMBLADO:  
HOLC YORLD  
---  
Liberando memoria... Sistema apagado.
```