Universidad Politécnica de Victoria.

Ingeniería en Tecnologías de la Información e Innovación Digital.

Materia: Estructura de Datos.

Docente: Dr. Said Polanco Martagón.

Unidad II.

Caso de Estudio: Decodificador de Protocolo Industrial (PRT-7).

Grupo: ITIID-76128

Alumna: Ana Zavala Arias. 2430118.

Cd. Victoria, Tamaulipas, a 06 de Noviembre de 2025.

Decodificador de Protocolo Industrial PRT-7

1. Introducción

El presente documento detalla el diseño, desarrollo e implementación del Decodificador de Protocolo Industrial PRT-7, un sistema de software en C++ diseñado para resolver un desafío de ciberseguridad industrial. La misión principal fue interceptar y descifrar la telemetría de un sensor (simulado por un dispositivo Arduino) que no transmite datos cifrados convencionalmente, sino instrucciones para la construcción de un mensaje oculto a través del protocolo PRT-7.

El desafío radica en que el mensaje se ensambla dinámicamente: las instrucciones de carga de caracteres (TramaLoad) interactúan con un "disco de cifrado" (RotorDeMapeo) cuyo estado es modificado por las instrucciones de mapeo (TramaMap). Esto requiere la gestión simultánea de estructuras de datos complejas (Listas Doblemente Enlazadas y Listas Circulares) y la aplicación de Polimorfismo y Comunicación Serial en tiempo real.

---

2. Manual Técnico

2.1 Diseño del Sistema (Arquitectura POO)

El diseño del decodificador se basa en una arquitectura de Programación Orientada a Objetos (POO) que utiliza Herencia y Polimorfismo para manejar los diferentes tipos de tramas recibidas.

2.1.1 Jerarquía de Clases (Polimorfismo)

Se define una interfaz común para todas las operaciones de procesamiento de tramas:

- **Clase Base Abstracta: TramaBase**
    - **Propósito:** Define el contrato para el procesamiento de cualquier trama.
    - **Método Virtual Puro:** `virtual void procesar(ListaDeCarga* carga, RotorDeMapeo* rotor) = 0;`. Esta es la clave del

polimorfismo, permitiendo que el código principal llame al método sin saber si es una carga o una rotación.

- **Destructor Virtual:** `virtual ~TramaBase() {}`. Este destructor es **esencial** para garantizar la correcta liberación de memoria (`delete`) de los objetos derivados a través del puntero de la clase base (`TramaBase*`), previniendo fugas de memoria.

- **Clase Derivada: TramaLoad**
  - **Representa:** Tramas `L,X` (carga de carácter).
  - **Función:** Decodifica su carácter interno (`caracter`) utilizando el estado actual del `RotorDeMapeo` y luego inserta el resultado en la `ListaDeCarga`.
- **Clase Derivada: TramaMap**
  - **Representa:** Tramas `M,N` (rotación del rotor).
  - **Función:** Simplemente invoca el método `rotar(int N)` del `RotorDeMapeo`, cambiando el estado de cifrado para las tramas futuras.

2.1.2 Estructuras de Datos (Implementación Manual)

Se requiere la implementación manual de dos listas doblemente enlazadas:

1. **`RotorDeMapeo` (Lista Circular Doblemente Enlazada):**
   - Simula un disco de cifrado tipo César/Enigma.
   - Se inicializa con los caracteres A-Z en un ciclo.
   - Contiene un puntero **`cabeza`** que define el punto de inicio del mapeo y es el único puntero que se mueve con la rotación.
   - **`rotar(int N)`:** Mueve el puntero `cabeza` hacia adelante (`N > 0`) o hacia atrás (`N < 0`) de manera eficiente, sin mover los datos de los nodos.
2. **`ListaDeCarga` (Lista Doblemente Enlazada):**
   - Almacena el mensaje decodificado en orden de inserción.

- Implementa el método **`insertarAlFinal(char dato)`** para mantener el mensaje en la secuencia correcta.

2.2 Desarrollo y Lógica de Decodificación

El código principal (`main.cpp`) se centra en tres áreas críticas: la inicialización, la comunicación serial (POSIX/Linux) y el bucle de procesamiento.

2.2.1 Comunicación Serial (`configurarSerial`, `leerLineaSerial`)

La conexión al dispositivo Arduino se maneja con librerías POSIX (`<fcntl.h>`, `<unistd.h>`, `<termios.h>`) debido al requisito de no usar APIs de alto nivel.

- **`configurarSerial(const char* puerto)`:** Abre el puerto (`open`), establece la velocidad a **B9600** (Baud Rate), configura el modo **8N1** (8 bits de datos, sin paridad, 1 bit de stop), y pone el puerto en **modo raw** para evitar procesamiento de caracteres, esencial para protocolos de bajo nivel.
- **`leerLineaSerial(...)`:** Lee carácter por carácter (`read(fd, &c, 1)`) hasta encontrar un terminador de línea (`\n` o `\r`), cumpliendo el requisito de parseo manual.

2.2.2 Parseo de Trama (`parsearLinea`)

Esta función es responsable de interpretar la cadena de texto recibida (e.g., `"L,A"` o `"M,-5"`) y crear el objeto polimórfico correcto.

1. Determina el **Tipo de Trama** (`L` o `M`) basándose en el primer carácter.
2. Para las tramas 'M', realiza la conversión manual del resto de la cadena a un entero (`int rotacion`), manejando explícitamente el signo negativo (`-`).
3. Retorna el objeto instanciado (e.g., `new TramaLoad('A')`) como un puntero `TramaBase*`.

2.2.3 Lógica de Mapeo (`RotorDeMapeo::getMapeo`)

Esta es la clave de la decodificación. El mapeo no es un simple desplazamiento César, sino un mapeo dinámico basado en la posición rotada del rotor.

1. Calcula la posición del carácter de entrada (`in`) con respecto a 'A' (`posicionDelCaracter = in - 'A'`).
2. Desde el puntero **`cabeza`** (que ya está rotado), avanza `posicionDelCaracter` nodos.
3. El `dato` del nodo final es el carácter decodificado. El mapeo es: **`Decodificado = Rotor[Cabeza + (Entrada - 'A')]`**.

Componente	Tipo de Estructura	Función Esencial	Requisito No Funcional
RotorDeMapeo	Lista Doblemente Enlazada Circular	Almacena el alfabeto (A-Z) y permite la rotación eficiente del punto de mapeo (cabeza).	Implementación manual sin STL.
ListaDeCarga	Lista Doblemente Enlazada	Ensambla y almacena el mensaje decodificado en el orden correcto.	Implementación manual sin STL y gestión de memoria (destructor).
TramaBase	Clase Base Abstracta	Define la interfaz procesar y asegura la **liberación segura de memoria** con su **Destructor Virtual**.	Destructor virtual **OBLIGATORIO** para prevenir fugas.

parsearLinea	Función C-style	Procesa cadenas de texto (char*), identifica el tipo de trama y extrae el valor (carácter o entero) sin usar std::string.	Parseo de cadenas **manual**.
configurarSerial	Función POSIX (termios.h)	Configura la comunicación a 9600 baudios y modo 8N1 en el puerto COM especificado por el usuario.	Uso exclusivo de librerías estándar POSIX para serial.

2.2.4 El Bucle Principal (main) y Flujo de Procesamiento

El corazón del sistema es el bucle infinito en la función `main`, que gestiona la secuencia de operaciones:

1. **Inicialización de Estructuras:** Se crean las instancias de las estructuras de datos principales: `ListaDeCarga` y `RotorDeMapeo`. El constructor de `RotorDeMapeo` crea automáticamente la **Lista Circular Doblemente Enlazada** con el alfabeto 'A' a 'Z'.
2. **Comunicación:** El programa solicita al usuario el path del puerto serial (ej. `/dev/ttyUSB0`) y utiliza `configurarSerial()` para establecer la conexión a **9600 baudios** en modo **8N1 raw**.
3. **Bucle de Lectura:** La función `leerLineaSerial()` bloquea el programa esperando datos y lee carácter por carácter hasta encontrar un terminador (`\n` o `\r`).
4. **Control de Transmisión:** Se manejan señales especiales: la línea que comienza con `'I'` indica el inicio de la transmisión, y la línea que contiene `"FIN"` provoca la salida del bucle principal.

5. **Parseo y Creación del Objeto Polimórfico:** La trama recibida (ej. `"L,A"`) se pasa a `parsearLinea()`, que **determina el tipo** (`'L'` o `'M'`) y **crea dinámicamente** el objeto correcto (`new TramaLoad(...)` o `new TramaMap(...)`). Este objeto se retorna como un puntero a la clase base `TramaBase*`.
6. **Procesamiento Polimórfico:** El método clave es la llamada: `trama->procesar(&miListaDeCarga, &miRotorDeMapeo)`.
    - Si el objeto es `TramaLoad`, se ejecuta su método `procesar`, que llama a `RotorDeMapeo::getMapeo` e inserta el resultado en `ListaDeCarga::insertarAlFinal`.
    - Si el objeto es `TramaMap`, se ejecuta su método `procesar`, que llama a `RotorDeMapeo::rotar(int N)`.
7. **Liberación de Memoria:** El objeto creado dinámicamente se libera inmediatamente con `delete trama;`. Gracias al **Destructor Virtual** en `TramaBase`, se llama al destructor de la clase derivada correcta (`TramaLoad` o `TramaMap`), previniendo una fuga de memoria.
8. **Resultado Final:** Al recibir `"FIN"`, se cierra el puerto (`close(fd)`) y se llama a `ListaDeCarga::imprimirMensaje()` para mostrar el texto decodificado.