Universidad Politécnica de Victoria

Ingeniería en Tecnologías de la Información e Innovación Digital

Decodificador de Protocolo Industrial PRT-7

Reporte Técnico

Alumno: César Euresti

Asignatura: Estructura de Datos

Fecha: 5 de noviembre de 2025

Índice

1 Introducción

El presente reporte documenta la implementación de un **Decodificador de Protocolo Industrial PRT-7**, un sistema complejo que integra conceptos fundamentales de Programación Orientada a Objetos y Estructuras de Datos en C++.

1.1 Contexto del Problema

La firma de ciberseguridad ha interceptado un protocolo de comunicación serial no encriptado proveniente de dispositivos Arduino. Sin embargo, el protocolo no envía mensajes directos, sino que implementa un mecanismo de *ensamblaje dinámico* donde dos tipos de tramas cooperan para revelar el mensaje oculto:

- **Tramas LOAD (L):** Transportan fragmentos de datos individuales (caracteres).
- **Tramas MAP (M):** Contienen instrucciones para reordenar y transformar los datos ya recibidos.

El desafío radica en que las operaciones MAP modifican dinámicamente el significado de cada fragmento, similar a un rotor de cifrado tipo Enigma, creando un sistema donde la estructura de datos y la lógica de procesamiento están intrínsecamente acopladas.

1.2 Objetivos

1. Implementar una arquitectura orientada a objetos que modele la heterogeneidad de tramas mediante herencia y polimorfismo.
2. Desarrollar estructuras de datos enlazadas manualmente: *Lista Doblemente Enlazada* y *Lista Circular*.
3. Integrar comunicación serial para leer datos del puerto COM.
4. Demostrar la gestión eficiente de memoria mediante el uso de punteros y *operador new/delete*.

2 Manual Técnico

2.1 Diseño Arquitectónico

El sistema implementa una arquitectura basada en **herencia polimórfica** que permite tratar diferentes tipos de tramas de manera uniforme a través de una interfaz común.

2.1.1 Jerarquía de Clases

La base del sistema es la clase abstracta `TramaBase`, que define el contrato que toda trama debe cumplir:

```
class TramaBase {
public:
    virtual void procesar(
   ListaDeCarga* carga,

   RotorDeMapeo* rotor) = 0;
    virtual ~TramaBase() {}
};
```

De esta clase heredan dos implementaciones concretas:

- **TramaLoad:** Encapsula una trama ‘L,X‘ donde ‘X‘ es un carácter. Su método ‘procesar()‘ inserta el carácter mapeado en la lista de carga.
- **TramaMap:** Encapsula una trama ‘M,N‘ donde ‘N‘ es un entero positivo o negativo. Su método ‘procesar()‘ rota el rotor de mapeo ‘N‘ posiciones.

2.1.2 Estructura: ListaDeCarga

La `ListaDeCarga` es una **Lista Doblemente Enlazada** que almacena los caracteres decodificados en el orden de llegada:

```
class ListaDeCarga {
private:
    struct Nodo {
        char dato;
        Nodo* siguiente;
        Nodo* anterior;
    };
    Nodo* cabeza;
```

```
public:
    void insertarAlFinal(char dato);
    void imprimirMensaje() const;
};
```

Operaciones principales:

- `insertarAlFinal(char)`: Añade un carácter al final de la lista, preservando el orden de llegada.
- `imprimirMensaje()`: Recorre la lista y despliega el mensaje completo decodificado.

2.1.3 Estructura: RotorDeMapeo

El `RotorDeMapeo` es una **Lista Circular Doblemente Enlazada** que actúa como un disco de cifrado (similar a una Rueda de César):

```
class RotorDeMapeo {
private:
    struct Nodo {
        char letra;
        Nodo* siguiente;
        Nodo* anterior;
    };
    Nodo* cabeza;
public:
    void rotar(int n);
    char getMapeo(char entrada);
};
```

Operaciones principales:

- `rotar(int n)`: Desplaza la posición de 'cabeza' circularmente 'n' posiciones. Los valores negativos rotan en dirección inversa.
- `getMapeo(char)`: Localiza el carácter de entrada en la lista, determina su distancia a 'cabeza', y devuelve el carácter mapeado correspondiente.

2.2 Componentes del Sistema

El sistema está compuesto por los siguientes módulos:

2.2.1 ArduinoParser

Este componente gestiona la comunicación con el puerto serial del Arduino:

- Abre la conexión con el dispositivo Arduino (típicamente '/dev/ttyACM0' en Linux).
- Lee líneas de texto del buffer del puerto.
- Proporciona métodos para verificar disponibilidad de datos.

2.2.2 LineaDispatcher

El dispatcher es el orquestador principal del sistema:

- Recibe cadenas del parser serial.
- Determina el tipo de trama (LOAD o MAP).
- Instancia los objetos 'TramaLoad' o 'TramaMap' correspondientes.
- Invoca el método 'procesar()' polimórfico.
- Gestiona la liberación de memoria.

2.2.3 Gestión de Memoria

La aplicación implementa **gestión manual de memoria** mediante 'new' y 'delete':

```
TramaBase* trama = nullptr;
if (tipo == 'L') {
    trama = new TramaLoad(caracter);
} else {
    trama = new TramaMap(numero);
}

trama->procesar(&listaCarga, &rotor);
delete trama;  // Memory liberation
```

La destrucción polimórfica es segura porque 'TramaBase' posee un destructor virtual.

2.3 Flujo de Procesamiento

El algoritmo de decodificación procede en los siguientes pasos:

1. **Inicialización:** Se crea la ‘ListaDeCarga‘ (vacía) y el ‘RotorDeMapeo‘ (A-Z, cabeza en ’A’).
2. **Lectura Serial:** El programa abre el puerto serial del dispositivo Arduino (ej. ‘/dev/ttyACM0‘ en Linux) y espera datos.
3. **Análisis de Tramas:** Para cada línea recibida:
   - Se parsea el formato: ‘L,X‘ o ‘M,N‘
   - Se identifica el tipo y se extrae el parámetro
4. **Procesamiento Polimórfico:** Se ejecuta el método ‘procesar()‘ apropiado:
   - **Para TramaLoad:** Se mapea el carácter a través del RotorDeMapeo y se inserta en la ListaDeCarga.
   - **Para TramaMap:** Se rota el RotorDeMapeo, alterando el estado del mapeo para futuras operaciones LOAD.
5. **Descodificación Final:** Una vez procesadas todas las tramas, se imprime el mensaje decodificado.

2.4 Ejemplo de Ejecución

Consideremos el siguiente flujo:

```
Entrada:    L,A    M,3    L,B    M,-1
      L,C
Rotor:      [A-Z]  Rota 3  Rota -1
   (...)
Salida:     Mensaje decodificado
```

Cuando llega ‘L,A‘:

- El RotorDeMapeo está en posición *cabeza* en ’A’
- Se busca ’A’ en el rotor, se encuentra en posición 0
- Se mapea el carácter y se inserta

Cuando llega ‘M,3‘:

- El RotorDeMapeo rota 3 posiciones: cabeza ahora apunta a ’D’
- El estado del mapeo cambia para futuras operaciones LOAD

2.5 Diagramas de Arquitectura

Esta subsección presenta los diagramas automáticos generados por Doxygen, que ilustran la arquitectura del sistema a nivel de implementación.

2.5.1 Jerarquía de Herencia

La siguiente imagen muestra la jerarquía de clases y las relaciones de herencia entre la clase base abstracta ‘TramaBase‘ y sus implementaciones concretas:

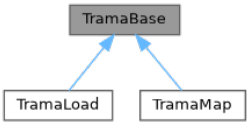

Figura 1: Diagrama de herencia de TramaBase

2.5.2 Grafo de Colaboraciones

Los siguientes diagramas muestran las colaboraciones y dependencias de cada clase especializada:

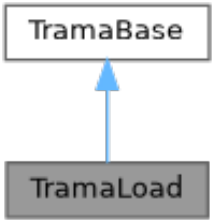

Figura 2: Colaboraciones de TramaLoad

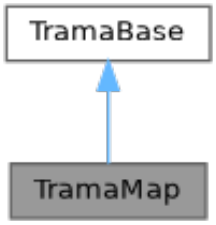

Figura 3: Colaboraciones de TramaMap

Estos diagramas revelan como cada tipo de trama interactúa con las estructuras de datos principales (ListaDeCarga y RotorDeMapeo) mediante sus métodos virtuales.

3 Capturas de Pantalla

Las siguientes imágenes documentan la ejecución del programa:

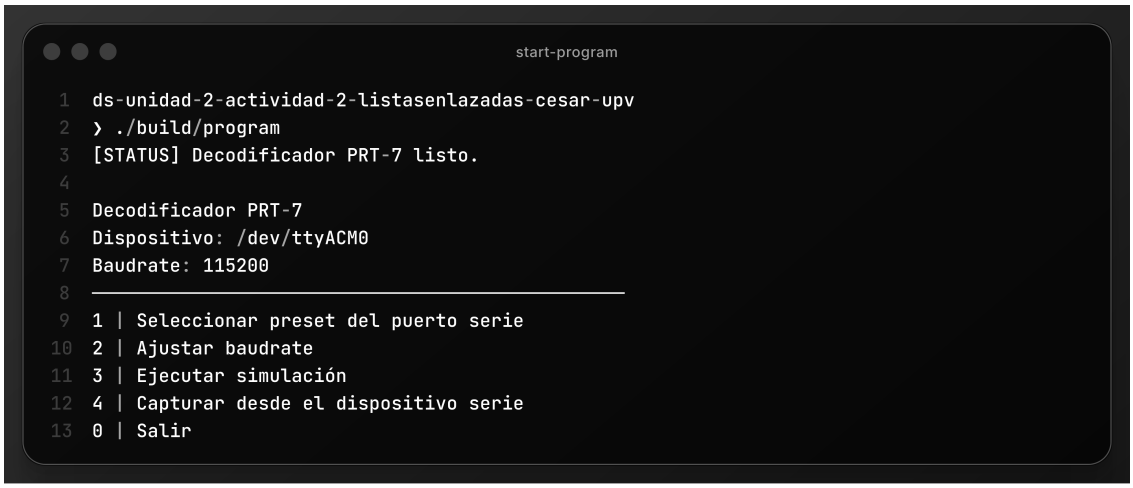

Figura 4: Inicio del programa y menu principal

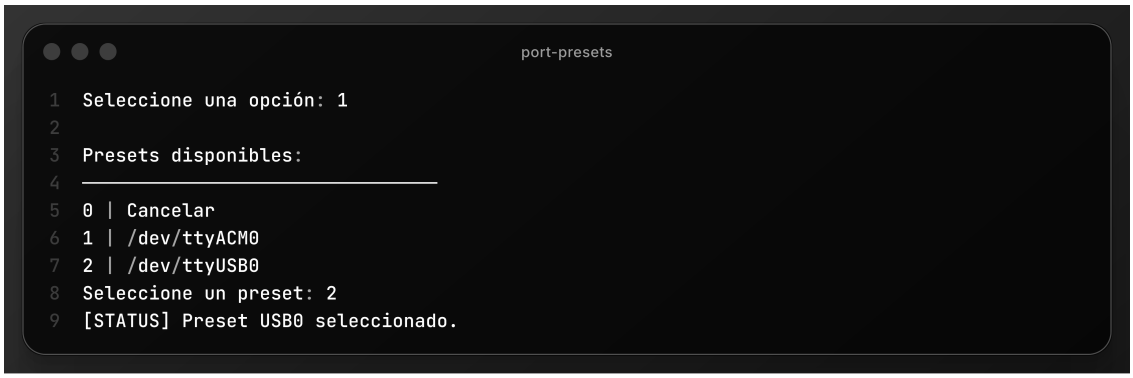

Figura 5: Configuracion de puertos preestablecidos

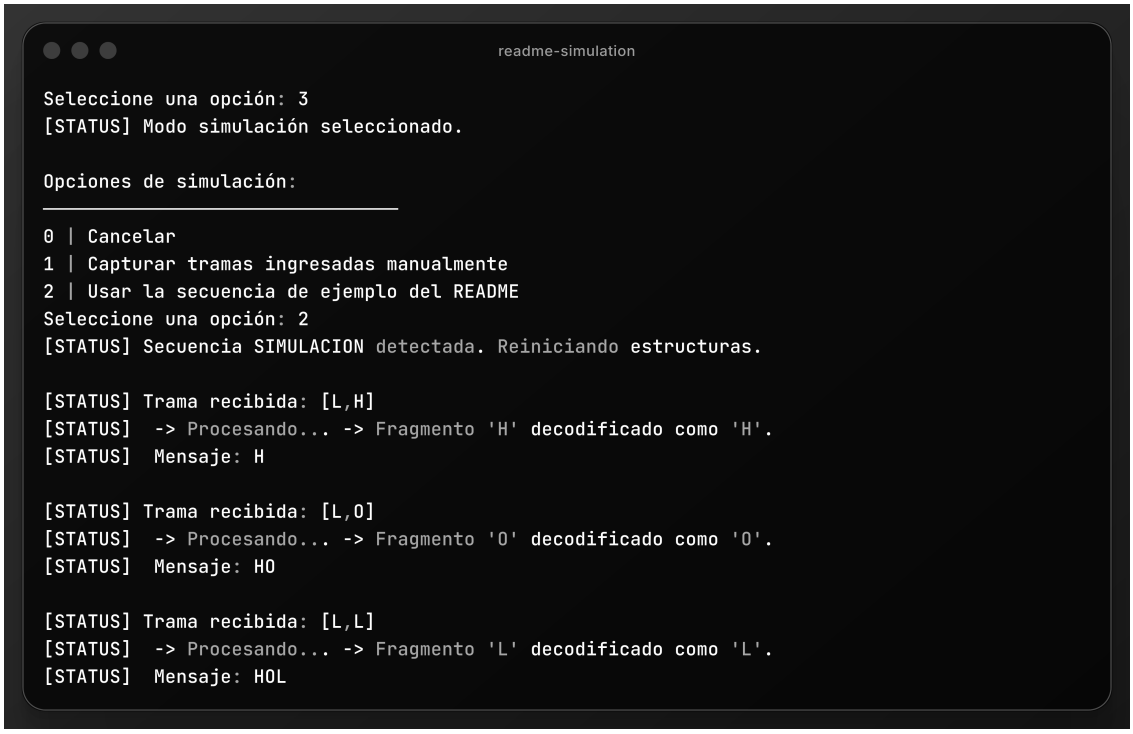

Figura 6: Simulacion del protocolo PRT-7

4 Conclusiones

La implementación del Decodificador PRT-7 demuestra la integración efectiva de:

- **Programación Orientada a Objetos:** Mediante herencia polimórfica y métodos virtuales.
- **Estructuras de Datos:** Listas enlazadas doblemente enlazadas y circulares, implementadas manualmente.
- **Gestión de Memoria:** Asignación dinámica segura con destructores virtuales.
- **Comunicación Serial:** Integración con dispositivos hardware reales.

Este proyecto es un ejemplo educativo valioso que prepara al ingeniero para sistemas embebidos complejos y comunicación de protocolos industriales.