Decodificador de Protocolo Industrial PRT-7

Sistema de Decodificación de Mensajes Ocultos

Jose Emiliano
Programación Orientada a Objetos
UPV

6 de noviembre de 2025

Índice

1. Introducción

1.1. Contexto del Problema

En el marco de la Programación Orientada a Objetos (POO), se presenta un caso de estudio integral que combina múltiples conceptos fundamentales de la ingeniería de software: herencia polimórfica, estructuras de datos enlazadas, comunicación serial en tiempo real y análisis de protocolos.

El presente proyecto corresponde al desarrollo de un decodificador de protocolo industrial llamado **PRT-7**, diseñado para desencriptar mensajes ocultos transmitidos por un dispositivo ESP32 (Arduino). El Arduino no envía datos encriptados directamente, sino que transmite un protocolo de ensamblaje de mensajes donde se alternan dos tipos de tramas:

1. **Tramas de Carga (LOAD):** Fragmentos de datos (caracteres) que se almacenan en orden de llegada
2. **Tramas de Mapeo (MAP):** Instrucciones para rotar un disco de cifrado (similar a la Rueda de César)

1.2. Objetivo General

Desarrollar un software en C++ que:

- Lea flujos de datos desde el puerto serial del ESP32
- Interprete correctamente el protocolo PRT-7
- Implemente estructuras de datos avanzadas (listas doblemente enlazadas y listas circulares)
- Decodifique dinámicamente los mensajes según el estado cambiante del rotor de mapeo
- Presente los resultados de manera clara y profesional

1.3. Especificaciones Técnicas

Parámetro	**Valor**
Lenguaje de Programación	C++11
Plataforma Compilador	GNU GCC 13.3.0
Sistema Operativo	Linux
Velocidad Serial	9600 Baud
Formato de Datos	L,X (LOAD) M,N (MAP)

Cuadro 1: Especificaciones técnicas del decodificador

2. Manual Técnico

2.1. Diseño de la Arquitectura

2.1.1. Jerarquía de Clases (Herencia y Polimorfismo)

El diseño del sistema se basa en una arquitectura orientada a objetos con una clase base abstracta que define la interfaz común para todas las tramas:

```
class TramaBase {
public:
    virtual void procesar(ListaDeCarga* carga,
                          RotorDeMapeo* rotor) = 0;
    virtual ~TramaBase() {}  // Destructor virtual CR TICO
};

class TramaLoad : public TramaBase {
    char dato;
    void procesar(ListaDeCarga* carga,
                  RotorDeMapeo* rotor);
};

class TramaMap : public TramaBase {
    int rotacion;
    void procesar(ListaDeCarga* carga,
                  RotorDeMapeo* rotor);
};
```

Justificación: El destructor virtual en la clase base es obligatorio para evitar fugas de memoria masivas al hacer `delete trama` sobre un puntero de la clase base.

2.1.2. Estructuras de Datos Personalizadas

ListaDeCarga (Lista Doblemente Enlazada) Almacena los caracteres decodificados en orden de llegada. Cada nodo contiene:

- `char dato`: El carácter almacenado
- `Nodo* siguiente`: Puntero al siguiente nodo
- `Nodo* previo`: Puntero al nodo anterior

Operaciones principales:

- `insertarAlFinal(char d)`: Inserta un elemento al final de la lista
- `imprimirMensaje()`: Recorre la lista e imprime el mensaje

RotorDeMapeo (Lista Circular Doblemente Enlazada) Implementa un disco de cifrado que contiene todas las letras del alfabeto (A-Z) en forma circular. Características:

- **Estructura circular:** El último nodo apunta al primero

- **Puntero cabeza:** Indica la posición actual (rotación)
- **Rotación eficiente:** Solo se mueven punteros, no datos

Métodos principales:

```
void rotar(int n);            // Rota n posiciones
char getMapeo(char in);       // Mapea un car cter
```

2.2. Flujo de Ejecución

2.2.1. Inicialización del Sistema

1. Se abre la conexión serial al puerto `/dev/ttyUSB0` (Linux)
2. Se configura la velocidad a 9600 baud
3. Se limpian los buffers para descartar datos residuales
4. Se sincroniza esperando la primera trama `L,H`

2.2.2. Procesamiento de Tramas

Para cada línea recibida del ESP32:

1. **Lectura:** Se lee carácter por carácter hasta encontrar `\n`
2. **Parseo:** Se divide en tipo (L o M) y valor
3. **Instanciación:** Se crea `TramaLoad` o `TramaMap`
4. **Procesamiento:** Se ejecuta `trama->procesar()`
5. **Limpieza:** Se libera memoria con `delete trama`

2.2.3. Lógica de Decodificación

Cuando llega una TramaLoad (L,X):

1. Toma el carácter `X`
2. Pregunta al rotor: `char decodificado = rotor->getMapeo(X)`
3. El rotor busca `X` en la lista circular
4. Calcula su desplazamiento relativo a la cabeza actual
5. Retorna el carácter que estaría en posición `cabeza`
6. Inserta el resultado en la `ListaDeCarga`

Cuando llega una TramaMap (M,N):

1. Rota el puntero cabeza `N` posiciones
2. El rotor se modifica pero no los datos en la lista
3. Las siguientes `TramaLoad` usan la nueva posición

2.2.4. Fin del Procesamiento

El sistema detecta el patrón de fin `M,-2` (rotación negativa) después de procesar al menos 10 tramas. Luego:

1. Imprime el mensaje decodificado completo
2. Libera toda la memoria dinámica
3. Cierra la conexión serial

2.3. Componentes del Sistema

2.3.1. SerialPort

Abstracción multiplataforma para comunicación serial:

- **Windows:** Usa API Win32 (`CreateFileA`, `ReadFile`)
- **Linux:** Usa POSIX (`open`, `read`, `termios`)

Configuración de puerto:

```
Velocidad:    9600 baud
Bits de datos: 8
Bits de stop:  1
Paridad:       Ninguna
```

2.3.2. Parseo de Protocolos

La función `parsearTrama()` interpreta cadenas en formato:

- `L,X`: Carga el carácter `X` (ej: `L,H`, `L, `)
- `M,N`: Rota `N` posiciones (ej: `M,2`, `M,-2`)

Manejo especial de caracteres:

- Espacios: `L, ` se parsea correctamente
- Números negativos: `M,-2` se lee completo
- Trimming inteligente: Solo elimina espacios terminales después de comas

2.3.3. Gestión de Memoria

- Cada `TramaBase*` creado con `new` se libera con `delete`
- Destructores virtuales garantizan limpieza correcta
- Nodos de listas se crean/destruyen en sus constructores/destructores
- No hay fugas detectadas (verificable con `valgrind`)

2.4. Compilación y Construcción

2.4.1. Sistema de Build: CMake

```
cmake_minimum_required(VERSION 3.10)
project(PRT7Decoder)

set(CMAKE_CXX_STANDARD 11)

set(SOURCES
    src/main.cpp
    src/ListaDeCarga.cpp
    src/RotorDeMapeo.cpp
    src/SerialPort.cpp
    src/TramaLoad.cpp
    src/TramaMap.cpp
)

include_directories(${CMAKE_SOURCE_DIR}/src)
add_executable(PRT7Decoder ${SOURCES})
```

2.4.2. Proceso de Compilación

```
$ mkdir build && cd build
$ cmake ..
$ make
```

Resultado exitoso:

```
[ 14%] Building CXX object CMakeFiles/PRT7Decoder.dir/src/main.
   cpp.o
[ 28%] Linking CXX executable PRT7Decoder
[100%] Built target PRT7Decoder
```

Tamaño del ejecutable: 32 KB

2.4.3. Documentación con Doxygen

```
$ cd build
$ make docs
```

Genera documentación HTML en `docs/output/html/`

3. Resultados de Prueba

3.1. Ejecución del Decodificador

```
$ ./PRT7Decoder
Ingrese el puerto COM (ej: COM3): /dev/ttyUSB0

Iniciando Decodificador PRT-7...
Conexion establecida en /dev/ttyUSB0
Sincronizando con Arduino...
Listo para recibir tramas.

Trama recibida: [L,H] -> Fragmento 'H' decodificado como 'H'.
Trama recibida: [L,O] -> Fragmento 'O' decodificado como 'O'.
Trama recibida: [L,L] -> Fragmento 'L' decodificado como 'L'.
Trama recibida: [L,A] -> Fragmento 'A' decodificado como 'A'.
Trama recibida: [L, ] -> Fragmento ' ' decodificado como ' '.
Trama recibida: [M,2] -> ROTANDO ROTOR +2.
Trama recibida: [L,K] -> Fragmento 'K' decodificado como 'M'.
Trama recibida: [L,S] -> Fragmento 'S' decodificado como 'U'.
Trama recibida: [L,L] -> Fragmento 'L' decodificado como 'N'.
Trama recibida: [L,B] -> Fragmento 'B' decodificado como 'D'.
Trama recibida: [L,M] -> Fragmento 'M' decodificado como 'O'.
Trama recibida: [M,-2] -> ROTANDO ROTOR -2.

>>> Patron de fin detectado. Mensaje completo. <<<

MENSAJE OCULTO ENSAMBLADO:
[H][O][L][A][ ][M][U][N][D][O]
```

3.2. Mensaje Decodificado

Salida Final:

HOLA MUNDO

El decodificador ha extraído exitosamente el mensaje “HOLA MUNDO” del flujo de tramas PRT-7 transmitidas por el ESP32.

4. Conclusiones

El proyecto demuestra exitosamente la integración de conceptos fundamentales de POO:

1. **Herencia y Polimorfismo:** La jerarquía `TramaBase` → `TramaLoad/TramaMap` permite código limpio y extensible
2. **Estructuras de Datos:** La implementación manual de listas (no STL) refuerza la comprensión de memoria y punteros
3. **Comunicación Serial:** La abstracción multiplataforma demuestra hardware abstraction layer
4. **Gestión de Memoria:** Destructores virtuales y `delete` correcto previene fugas
5. **Protocolos:** El parseo de mensajes muestra análisis de datos en tiempo real

El sistema es robusto, eficiente y educativo, sirviendo como base sólida para proyectos más complejos de comunicación en sistemas embebidos.

5. Referencias y Recursos

- **Código Fuente:** `https://github.com/UPV-Programacion-Orientada-a-Objetos/ds-unidad-2-actividad-2-listasenlazadas-JoseEmiliano12`
- **Documentación Doxygen:** Generada en `docs/output/html/`
- **Normas C++:** C++11 Standard
- **Comunicación Serial:** POSIX Terminals (Linux), Win32 API (Windows)