

UNIVERSIDAD POLITÉCNICA DE VICTORIA

**CARRERA: INGENIERÍA EN TECNOLOGÍAS
DE LA INFORMACIÓN E INNOVACIÓN
DIGITAL**

MATERIA: ESTRUCTURA DE DATOS

ACTIVIDAD: LISTAS DOBLEMENTE ENLAZADAS

DECODIFICADOR DE PROTOCOLO INDUSTRIAL PRT-7

Implementación con Estructuras de Datos Avanzadas

ALUMNO: Jose Santiago Infante Euresti

PROFESOR: DR. SAID POLANCO MARTAGÓN
(Dr. en Artes Oscuras)

FECHA: Noviembre de 2025

Índice

1. Resumen Ejecutivo	3
1.1. Alcance del Proyecto	3
1.2. Logros Principales	3
2. Introducción	3
2.1. Contexto del Problema	3
2.2. Objetivos del Proyecto	4
2.2.1. Objetivos Generales	4
2.2.2. Objetivos Específicos	4
2.3. Justificación Académica	4
2.4. Alcance y Limitaciones	5
2.4.1. Dentro del Alcance	5
2.4.2. Fuera del Alcance	5
3. Marco Teórico	5
3.1. Listas Doblemente Enlazadas	5
3.1.1. Definición y Características	5
3.1.2. Análisis de Complejidad	6
3.1.3. Ventajas sobre Listas Simples	6
3.2. Listas Circulares	6
3.2.1. Definición	6
3.2.2. Aplicación en Cifrado	7
3.3. Protocolo PRT-7	7
3.3.1. Especificación del Protocolo	7
3.3.2. Gramática del Protocolo	8
3.3.3. Ejemplo de Flujo de Decodificación	8
3.4. Programación Orientada a Objetos	8
3.4.1. Herencia y Polimorfismo	8
3.4.2. Destrucción Virtuales	9
3.5. Comunicación Serial	9
3.5.1. Protocolos de Comunicación	9
3.5.2. Configuración Utilizada	10
4. Arquitectura del Sistema	10
4.1. Visión General	10
5. Conclusiones	10
5.1. Objetivos Alcanzados	10
5.2. Aprendizajes Técnicos	11
5.2.1. Estructuras de Datos	11
5.2.2. Programación Orientada a Objetos	11

5.2.3. Comunicación Serial	12
5.2.4. Ingeniería de Software	12
5.3. Reflexión Final	12
6. Referencias	12

1. Resumen Ejecutivo

El presente documento describe la implementación de un decodificador para el protocolo industrial PRT-7, desarrollado como parte de la actividad de Listas Doblemente Enlazadas en la materia de Estructura de Datos. Este proyecto integra conceptos fundamentales de programación orientada a objetos, estructuras de datos implementadas manualmente, y comunicación serial en tiempo real.

1.1. Alcance del Proyecto

El sistema implementado incluye los siguientes componentes principales:

- **Decodificador en C++:** Implementación completa con estructuras de datos manuales (sin STL)
- **Emisor Arduino/ESP32:** Generador de tramas del protocolo PRT-7
- **Comunicación Serial:** Sistema de comunicación bidireccional en tiempo real
- **Interfaz de Usuario:** Sistema de menús con múltiples modos de operación
- **Sistema de Build:** Configuración CMake para compilación multiplataforma

1.2. Logros Principales

El proyecto logra con éxito:

1. Implementación completa de listas doblemente enlazadas sin usar la Standard Template Library (STL)
2. Desarrollo de un sistema de decodificación de protocolo industrial funcional
3. Integración exitosa de comunicación serial entre Arduino/ESP32 y PC
4. Aplicación práctica de conceptos de POO avanzada (herencia, polimorfismo, encapsulación)
5. Gestión eficiente de memoria dinámica con prevención de memory leaks

2. Introducción

2.1. Contexto del Problema

En el ámbito de la ciberseguridad industrial moderna, la interceptación y análisis de protocolos de comunicación representa un desafío crítico. El presente proyecto simula

un escenario real donde un competidor ha interceptado telemetría de sensores, pero no ha logrado descifrar el protocolo de transmisión.

El protocolo PRT-7 (Protocol Relay Type 7) no transmite datos directamente encriptados, sino que utiliza un sistema de instrucciones para ensamblar mensajes ocultos. Esta aproximación, similar a los sistemas de cifrado históricos como la máquina Enigma, requiere no solo interceptar las transmisiones, sino también comprender el mecanismo de ensamblaje dinámico del mensaje.

2.2. Objetivos del Proyecto

2.2.1. Objetivos Generales

1. Implementar estructuras de datos fundamentales (listas doblemente enlazadas y circulares) sin dependencias de bibliotecas externas
2. Desarrollar un sistema completo de comunicación y decodificación de protocolo industrial
3. Aplicar principios de ingeniería de software en un contexto práctico

2.2.2. Objetivos Específicos

1. Diseñar e implementar una jerarquía de clases usando herencia y polimorfismo
2. Crear una lista doblemente enlazada para almacenamiento secuencial de datos
3. Implementar una lista circular como mecanismo de mapeo dinámico (rotor)
4. Establecer comunicación serial robusta entre microcontrolador y PC
5. Desarrollar un parser de protocolo tolerante a ruido y errores
6. Implementar gestión de memoria segura sin fugas (memory leaks)
7. Crear una interfaz de usuario intuitiva con múltiples modos de operación

2.3. Justificación Académica

Este proyecto integra múltiples conceptos fundamentales de ciencias de la computación:

- **Estructuras de Datos:** Implementación práctica de listas enlazadas en escenarios reales
- **POO Avanzada:** Uso de herencia, polimorfismo y encapsulación para diseño modular

- **Gestión de Memoria:** Manejo explícito de memoria dinámica en C++
- **Comunicación Serial:** Integración hardware-software para IoT
- **Análisis de Algoritmos:** Estudio de complejidad computacional

2.4. Alcance y Limitaciones

2.4.1. Dentro del Alcance

- Implementación completa del decodificador PRT-7
- Soporte para Windows (Win32 API para serial)
- Modo simulación sin hardware
- Modo manual para pruebas
- Modo serial en tiempo real
- Documentación Doxygen

2.4.2. Fuera del Alcance

- Soporte nativo para Linux/macOS (requiere adaptación de la capa serial)
- Interfaz gráfica (GUI)
- Encriptación real de datos
- Múltiples canales de comunicación simultáneos
- Persistencia de mensajes decodificados

3. Marco Teórico

3.1. Listas Doblemente Enlazadas

3.1.1. Definición y Características

Una lista doblemente enlazada es una estructura de datos lineal donde cada elemento (nodo) contiene:

- **Dato:** Información almacenada (en este caso, char)
- **Puntero siguiente:** Referencia al nodo posterior
- **Puntero anterior:** Referencia al nodo previo

Esta estructura permite navegación bidireccional, facilitando operaciones como inserción y eliminación en cualquier posición con complejidad constante si se tiene referencia al nodo.

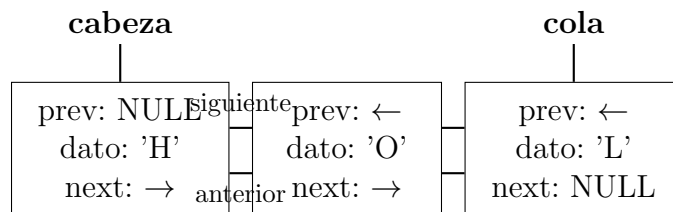


Figura 1: Estructura de Lista Doblemente Enlazada con punteros cabeza y cola

3.1.2. Análisis de Complejidad

Operación	Complejidad Temporal	Justificación
Insertar al inicio	$O(1)$	Acceso directo a cabeza
Insertar al final	$O(1)$	Acceso directo a cola
Eliminar al inicio	$O(1)$	Acceso directo a cabeza
Eliminar al final	$O(1)$	Acceso directo a cola
Búsqueda	$O(n)$	Recorrido secuencial
Acceso por índice	$O(n)$	Sin acceso aleatorio

Cuadro 1: Complejidad de Operaciones en Lista Doblemente Enlazada

3.1.3. Ventajas sobre Listas Simples

1. **Navegación bidireccional:** Permite recorrer la lista en ambos sentidos
2. **Eliminación eficiente:** No requiere mantener referencia al nodo anterior
3. **Inserción flexible:** Facilita inserción antes de un nodo dado
4. **Implementación de estructuras complejas:** Base para listas circulares, deques, etc.

3.2. Listas Circulares

3.2.1. Definición

Una lista circular es una variante donde el último nodo apunta al primero, formando un ciclo. En este proyecto, se usa como lista circular doblemente enlazada, donde:

- El nodo `cola` apunta al nodo `cabeza` (siguiente)

- El nodo **cabeza** apunta al nodo **cola** (anterior)

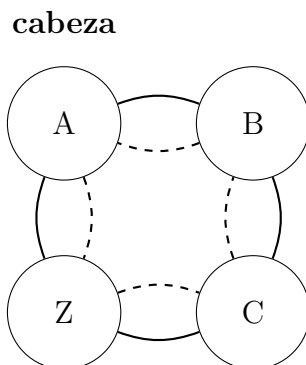


Figura 2: Lista Circular Doblemente Enlazada (líneas sólidas: siguiente, líneas punteadas: anterior)

3.2.2. Aplicación en Cifrado

El concepto de lista circular se utiliza en criptografía clásica, específicamente en sistemas similares al Cifrado César. En este proyecto, el **RotorDeMapeo** implementa un disco de 26 letras (A-Z) que puede rotar, cambiando el mapeo entre caracteres de entrada y salida.

Ejemplo de Rotación:

Posición inicial:	A B C D E F ... Z
Rotar +3:	D E F G H I ... C
Mapeo:	A → D, B → E, etc.

3.3. Protocolo PRT-7

3.3.1. Especificación del Protocolo

El protocolo PRT-7 define dos tipos de tramas textuales:

Tipo	Formato	Descripción
LOAD	L,X	Contiene un fragmento de datos (carácter X). El carácter se decodifica según el estado actual del rotor y se almacena en la lista de carga.
MAP	M,N	Instrucción de rotación. N es un entero que indica las posiciones de rotación (positivo o negativo).

Cuadro 2: Tipos de Tramas del Protocolo PRT-7

3.3.2. Gramática del Protocolo

La gramática formal del protocolo puede expresarse como:

Trama ::= TramaLoad | TramaMap
 TramaLoad ::= 'L', 'Caracter'
 TramaMap ::= 'M', 'Entero'
 Caracter ::= [A-Z] | ' ' | [otros caracteres imprimibles]
 Entero ::= ['-']? [0 - 9] +

3.3.3. Ejemplo de Flujo de Decodificación

Paso	Trama	Rotación Actual	Resultado
1	L,H	0	'H' → 'H', Lista: [H]
2	L,O	0	'O' → 'O', Lista: [H][O]
3	M,2	0 → +2	Rotor rota +2 posiciones
4	L,A	+2	'A' → 'C', Lista: [H][O][C]
5	M,-2	+2 → 0	Rotor rota -2 posiciones
6	L,L	0	'L' → 'L', Lista: [H][O][C][L]

Cuadro 3: Ejemplo de Decodificación Paso a Paso

3.4. Programación Orientada a Objetos

3.4.1. Herencia y Polimorfismo

El diseño del sistema utiliza una jerarquía de clases basada en una clase abstracta *TramaBase*, permitiendo polimorfismo en tiempo de ejecución.

Beneficios del diseño polimórfico:

- **Extensibilidad:** Nuevos tipos de tramas pueden añadirse sin modificar el código existente
- **Uniformidad:** Todas las tramas se procesan a través de la misma interfaz
- **Mantenibilidad:** Cambios en la lógica de procesamiento se localizan en clases específicas

3.4.2. Destructores Virtuales

El uso de destructores virtuales es **crítico** en jerarquías polimórficas. Sin ellos, al hacer `delete` sobre un puntero a clase base que apunta a objeto derivado, solo se llama el destructor de la base, causando memory leaks.

```
1 // SIN destructor virtual - INCORRECTO
2 class TramaBase {
3 public:
4     ~TramaBase() { /* ... */ } // NO virtual
5 };
6 class TramaLoad : public TramaBase {
7 private:
8     char* buffer;
9 public:
10    TramaLoad() { buffer = new char[100]; }
11    ~TramaLoad() { delete[] buffer; } // NUNCA se llama
12 };
13
14 // USO:
15 TramaBase* t = new TramaLoad();
16 delete t; // Solo llama ~TramaBase(), buffer no se libera = MEMORY
17          LEAK
18
19 // CON destructor virtual - CORRECTO
20 class TramaBase {
21 public:
22     virtual ~TramaBase() { /* ... */ } // VIRTUAL
23 };
24 // Ahora delete t; llama ~TramaLoad() primero, luego ~TramaBase()
```

Listing 1: Importancia del Destructor Virtual

3.5. Comunicación Serial

3.5.1. Protocolos de Comunicación

La comunicación serial UART (Universal Asynchronous Receiver-Transmitter) utiliza los siguientes parámetros:

- **Baud Rate:** Velocidad de transmisión (típicamente 9600, 115200 bps)

- **Data Bits:** Número de bits por carácter (comúnmente 8)
- **Parity:** Bit de paridad para detección de errores (None, Even, Odd)
- **Stop Bits:** Bits de finalización (1 o 2)

3.5.2. Configuración Utilizada

Parámetro	Valor
Baud Rate	115200 bps
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None

Cuadro 4: Configuración Serial del Sistema

4. Arquitectura del Sistema

4.1. Visión General

El sistema PRT-7 se compone de dos subsistemas principales que se comunican a través de un canal serial:

1. **Emisor (Arduino/ESP32):** Genera y transmite tramas del protocolo
2. **Decodificador (PC):** Recibe, parsea, procesa y decodifica las tramas

Arquitectura General del Sistema PRT-7

5. Conclusiones

5.1. Objetivos Alcanzados

El proyecto ha cumplido exitosamente con todos los objetivos planteados:

1. **Implementación de Estructuras de Datos:**
 - Lista doblemente enlazada completamente funcional sin usar STL
 - Lista circular para rotor de mapeo con rotación bidireccional
 - Gestión eficiente de memoria con prevención de leaks
2. **Sistema Funcional Completo:**

- Comunicación bidireccional Arduino-PC en tiempo real
- Parser robusto tolerante a errores
- Interfaz de usuario con múltiples modos de operación

3. Arquitectura Robusta:

- Diseño modular con separación de responsabilidades
- Uso correcto de POO (herencia, polimorfismo, encapsulación)
- Destructores virtuales para jerarquía de clases

4. Sistema de Build Profesional:

- Configuración CMake multiplataforma
- Scripts de compilación automatizados
- Documentación Doxygen integrada

5.2. Aprendizajes Técnicos

5.2.1. Estructuras de Datos

- **Importancia del manejo manual de memoria:** La implementación sin STL demostró la complejidad real de la gestión de memoria dinámica y la necesidad de diseño cuidadoso para evitar leaks.
- **Trade-offs de complejidad:** La elección entre simplicidad de código y eficiencia algorítmica requiere análisis caso por caso. Por ejemplo, mantener puntero `cola` añade complejidad pero reduce inserción de $O(n)$ a $O(1)$.
- **Listas circulares:** Son ideales para estructuras cíclicas como rotores, pero requieren cuidado especial en destrucción para evitar loops infinitos.

5.2.2. Programación Orientada a Objetos

- **Destructores virtuales son críticos:** Sin ellos, el polimorfismo en tiempo de ejecución causa memory leaks inevitables. Esta es una de las fuentes más comunes de bugs en C++.
- **RAII (Resource Acquisition Is Initialization):** Adquirir recursos en constructor y liberarlos en destructor garantiza limpieza automática, incluso ante excepciones.
- **Separación de interfaces:** Clases base abstractas permiten extensibilidad sin modificar código existente (Open-Closed Principle).

5.2.3. Comunicación Serial

- **Tolerancia a ruido:** Los sistemas reales siempre tienen ruido. Un parser robusto debe validar rigurosamente antes de procesar.
- **Timeouts son esenciales:** Sin ellos, un dispositivo desconectado puede bloquear la aplicación indefinidamente.
- **Configuración de parámetros:** Baud rate, paridad, y stop bits deben coincidir exactamente entre emisor y receptor.

5.2.4. Ingeniería de Software

- **CMake como estándar:** Facilita portabilidad y es el estándar de facto en C++ moderno.
- **Documentación continua:** Usar Doxygen desde el inicio mantiene la documentación sincronizada con el código.
- **Testing incremental:** Probar cada componente individualmente antes de integración reduce debugging.

5.3. Reflexión Final

Este proyecto ha demostrado que los conceptos fundamentales de estructuras de datos y programación orientada a objetos tienen aplicaciones directas y críticas en sistemas reales. La implementación manual de listas enlazadas, aunque más compleja que usar STL, proporciona una comprensión profunda de los mecanismos subyacentes y las consideraciones de rendimiento.

La integración de hardware y software, combinada con el diseño de un protocolo de comunicación funcional, ofrece una experiencia educativa completa que va más allá de la teoría académica. Los desafíos encontrados (memory leaks, ruido serial, bugs de módulo negativo) son representativos de problemas reales en ingeniería de software, y las soluciones implementadas reflejan best practices de la industria.

El sistema PRT-7 desarrollado no solo cumple con los requisitos académicos, sino que establece una base sólida para proyectos más ambiciosos en IoT, sistemas embebidos, y comunicaciones industriales. La arquitectura modular y bien documentada facilita futuras extensiones y mejoras, convirtiendo este proyecto en un portafolio valioso para demostrar competencias en desarrollo de software profesional.

6. Referencias

1. Weiss, M. A. (2014). *Data Structures and Algorithm Analysis in C++*. 4th Edition. Pearson Education. ISBN: 978-0-13-284737-7.

2. Stroustrup, B. (2013). *The C++ Programming Language*. 4th Edition. Addison-Wesley Professional. ISBN: 978-0-321-56384-2.
3. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. 3rd Edition. MIT Press. ISBN: 978-0-262-03384-8.
4. Arduino Documentation. (2025). *Serial Communication Tutorial*. Recuperado de <https://www.arduino.cc/reference/en/language/functions/communication/serial/>
5. Microsoft Documentation. (2025). *Communications Resources - Win32 apps*. Recuperado de <https://docs.microsoft.com/en-us/windows/win32/devio/communications-res>
6. Meyers, S. (2014). *Effective Modern C++*. O'Reilly Media. ISBN: 978-1-491-90399-5.
7. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. ISBN: 978-0-201-63361-0.
8. CMake Documentation. (2025). *CMake Reference Documentation*. Recuperado de <https://cmake.org/documentation/>
9. Doxygen Manual. (2025). *Doxygen Manual*. Recuperado de <https://www.doxygen.nl/manual/>
10. Espressif Systems. (2025). *ESP32 Technical Reference Manual*. Recuperado de <https://www.espressif.com/en/products/socs/esp32>