Universidad Autonoma de
Aguascalientes

Centro de Ciencias Basicas
Departamento de Ciencias de la Computacion

# Decodificador de Protocolo Industrial PRT-7

*Sistema de Decodificacion en Tiempo Real*

**Materia:** Estructuras de Datos

**Actividad:** Unidad 2 - Listas Enlazadas

**Caso de Estudio:** PRT-7

**Estudiante:**
Fabian Ramirez (FabiRamiro)

**Profesor:**
[Nombre del Profesor]

Enero 2025

# Índice


**1 Introduccion** **4**
- 1.1 Contexto del Proyecto . . . 4
- 1.2 Problema a Resolver . . . 4
- 1.3 Objetivos . . . 4
  - 1.3.1 Objetivo General . . . 4
  - 1.3.2 Objetivos Especificos . . . 4
- 1.4 Alcance . . . 5
- 1.5 Justificacion Academica . . . 5

**2 Manual Tecnico** **6**
- 2.1 Diseño del Sistema . . . 6
  - 2.1.1 Arquitectura General . . . 6
  - 2.1.2 Jerarquia de Clases . . . 6
  - 2.1.3 Protocolo PRT-7 . . . 7
- 2.2 Desarrollo . . . 9
  - 2.2.1 Estructura de Datos: RotorDeMapeo . . . 9
  - 2.2.2 Estructura de Datos: ListaDeCarga . . . 11
  - 2.2.3 Comunicacion Serial . . . 13
  - 2.2.4 Algoritmo de Decodificacion . . . 15
- 2.3 Componentes del Sistema . . . 18
  - 2.3.1 Modulo de Tramas . . . 18
  - 2.3.2 Modulo de Estructuras de Datos . . . 18
  - 2.3.3 Modulo de Comunicacion . . . 19
  - 2.3.4 Programa Principal . . . 20
  - 2.3.5 Sistema de Compilacion . . . 20
  - 2.3.6 Documentacion con Doxygen . . . 23
  - 2.3.7 Gestion de Memoria . . . 25

**3 Pantallazos de la Implementacion** **27**
- 3.1 Arduino - Codigo Emisor . . . 27
- 3.2 Monitor Serial de Arduino . . . 28
- 3.3 Ejecucion del Decodificador . . . 28
- 3.4 Prueba de Integracion Completa . . . 29
- 3.5 Documentacion Doxygen Generada . . . 30
- 3.6 Herramientas de Desarrollo . . . 30

**4 Conclusiones** **31**
- 4.1 Logros Alcanzados . . . 31
- 4.2 Desafios Enfrentados y Soluciones . . . 31
- 4.3 Aprendizajes Clave . . . 32
- 4.4 Aplicaciones Reales . . . 32
- 4.5 Trabajo Futuro . . . 33
- 4.6 Reflexion Final . . . 33

**5 Referencias** **34**

**A Fragmentos de Codigo Fuente Relevantes** **35**
A.1 TramaBase.h - Clase Abstracta . . . . . 35
A.2 Algoritmo de Parseo . . . . . 35

**B Instrucciones de Instalacion** **36**
B.1 Requisitos del Sistema . . . . . 36
B.2 Pasos de Instalacion . . . . . 37

# Índice de figuras


1 Estructura de archivos del proyecto . . . . . 6
2 Codigo fuente de TramaBase.h . . . . . 7
3 Codigo fuente de RotorDeMapeo.h . . . . . 9
4 Codigo fuente de ListaDeCarga.cpp . . . . . 12
5 Codigo fuente de SerialPort.h . . . . . 14
6 Codigo fuente de main.cpp . . . . . 17
7 Codigo fuente de TramaLoad.cpp . . . . . 18
8 Proceso de compilacion con CMake . . . . . 21
9 Compilacion con MinGW Make . . . . . 22
10 Pagina principal de la documentacion Doxygen . . . . . 23
11 Lista de clases en Doxygen . . . . . 24
12 Codigo del emisor PRT-7 en Arduino IDE . . . . . 27
13 Proceso de subida del codigo al Arduino . . . . . 27
14 Monitor serial mostrando las tramas transmitidas . . . . . 28
15 Pantalla de inicio del decodificador . . . . . 28
16 Procesamiento de tramas en tiempo real . . . . . 28
17 Mensaje final decodificado . . . . . 29
18 Sesion completa de decodificacion de inicio a fin . . . . . 29
19 Lista de archivos documentados en Doxygen . . . . . 30
20 Proyecto abierto en Visual Studio Code . . . . . 30

# 1 Introduccion

## 1.1 Contexto del Proyecto

El presente proyecto consiste en el desarrollo de un sistema de decodificacion del protocolo industrial PRT-7, utilizado para descifrar telemetria de sensores interceptados. El protocolo PRT-7 es un sistema de ensamblaje de mensajes ocultos que no transmite datos encriptados directamente, sino las *instrucciones* para construir el mensaje.

Este proyecto se desarrolla en el marco del curso de Estructuras de Datos, aplicando conceptos fundamentales como listas enlazadas, polimorfismo y gestion manual de memoria en C++.

## 1.2 Problema a Resolver

En un escenario de ciberseguridad industrial, se ha descubierto que un competidor intercepta la telemetria de sensores simulados por un dispositivo Arduino. Sin embargo, los datos interceptados no tienen sentido directo, ya que el sistema utiliza un protocolo de mapeo rotatorio que transforma los caracteres de manera dinamica.

El desafio principal es implementar un decodificador que:

- Lea tramas desde un puerto serial en tiempo real
- Interprete dos tipos de instrucciones (LOAD y MAP)
- Utilice estructuras de datos manuales (sin STL)
- Implemente polimorfismo para procesamiento de tramas
- Gestione memoria dinamica correctamente
- Detecte automaticamente el fin del mensaje

## 1.3 Objetivos

### 1.3.1. Objetivo General

Desarrollar un sistema de decodificacion en tiempo real del protocolo PRT-7 utilizando conceptos avanzados de Programacion Orientada a Objetos y estructuras de datos implementadas manualmente.

### 1.3.2. Objetivos Especificos

1. Implementar una jerarquia de clases polimorficas para el procesamiento de tramas
2. Desarrollar una lista circular doblemente enlazada para el rotor de mapeo
3. Crear una lista doblemente enlazada para almacenamiento secuencial de datos
4. Establecer comunicacion serial bidireccional con dispositivo Arduino

5. Validar el correcto funcionamiento mediante pruebas de integracion
6. Generar documentacion tecnica completa usando Doxygen
7. Implementar un sistema de compilacion redistribuible mediante CMake

## 1.4 Alcance

El proyecto abarca:

- Desarrollo completo en C++ sin uso de STL (Standard Template Library)
- Implementacion para plataforma Windows
- Comunicacion serial a 9600 baudios
- Soporte para alfabeto ingles (A-Z), mayusculas, minusculas y espacios
- Sistema de compilacion multiplataforma mediante CMake
- Documentacion automatizada con Doxygen
- Deteccion inteligente de patrones de finalizacion

## 1.5 Justificacion Academica

Este proyecto integra multiples conceptos fundamentales de Estructuras de Datos:

**Listas Enlazadas:** Implementacion manual de listas doblemente enlazadas y circulares sin uso de contenedores de la STL

**Polimorfismo:** Uso de clases abstractas y metodos virtuales para diseño extensible y mantenible

**Gestion de Memoria:** Manejo explicito de memoria dinamica con operadores new/-delete y destructores virtuales

**Integracion de Sistemas:** Comunicacion con hardware externo en tiempo real mediante API de Windows

**Herramientas Profesionales:** Uso de CMake, Doxygen y control de versiones (Git/GitHub)

# 2 Manual Tecnico

## 2.1 Diseño del Sistema

### 2.1.1. Arquitectura General

El sistema se estructura en tres capas principales que permiten una separacion clara de responsabilidades:

1. **Capa de Comunicacion:** Maneja la interfaz serial con el Arduino mediante la API de Windows
2. **Capa de Procesamiento:** Interpreta y ejecuta las tramas recibidas usando polimorfismo
3. **Capa de Datos:** Almacena y transforma la informacion usando estructuras enlazadas

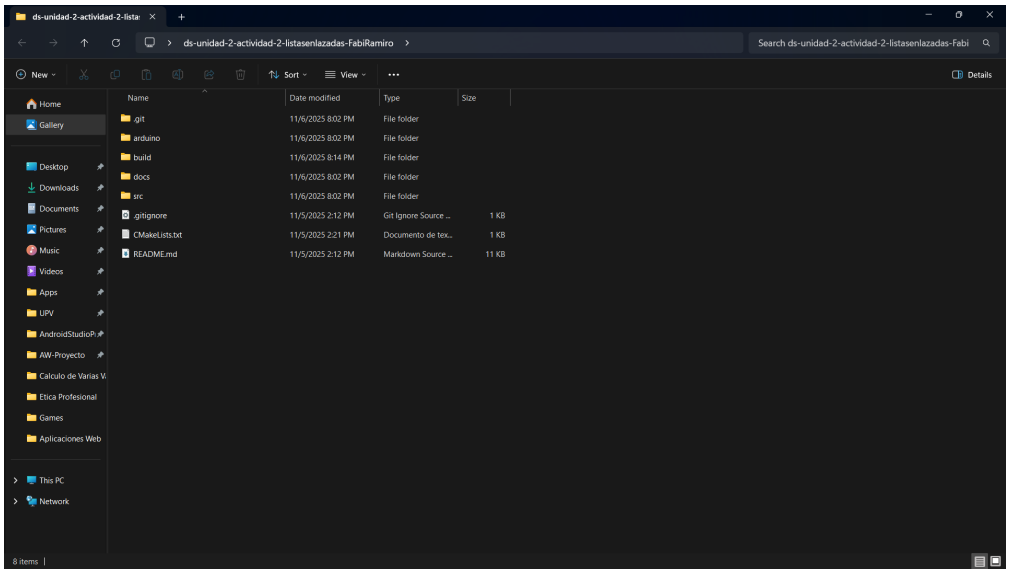



Figura 1: Estructura de archivos del proyecto

### 2.1.2. Jerarquia de Clases

El diseño orientado a objetos se basa en una jerarquia polimórfica que permite el procesamiento uniforme de diferentes tipos de tramas:

```
TramaBase (Clase Abstracta)
    |
    +-- TramaLoad  (Trama de carga de datos)
    |
    +-- TramaMap   (Trama de rotacion del rotor)
```

Listing 1: Jerarquia de clases principal

**Clase Base: TramaBase**

La clase base define la interfaz comun que todas las tramas deben implementar:

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

Listing 2: Definicion de TramaBase

El metodo `procesar()` es virtual puro ($= 0$), lo que obliga a las clases derivadas a proporcionar su propia implementacion. El destructor virtual es critico para evitar fugas de memoria al eliminar objetos a traves de punteros a la clase base.

**Clases Derivadas:**

- `TramaLoad`: Almacena un caracter y lo decodifica usando el estado actual del rotor
- `TramaMap`: Almacena un entero N y rota el rotor N posiciones (positivas o negativas)

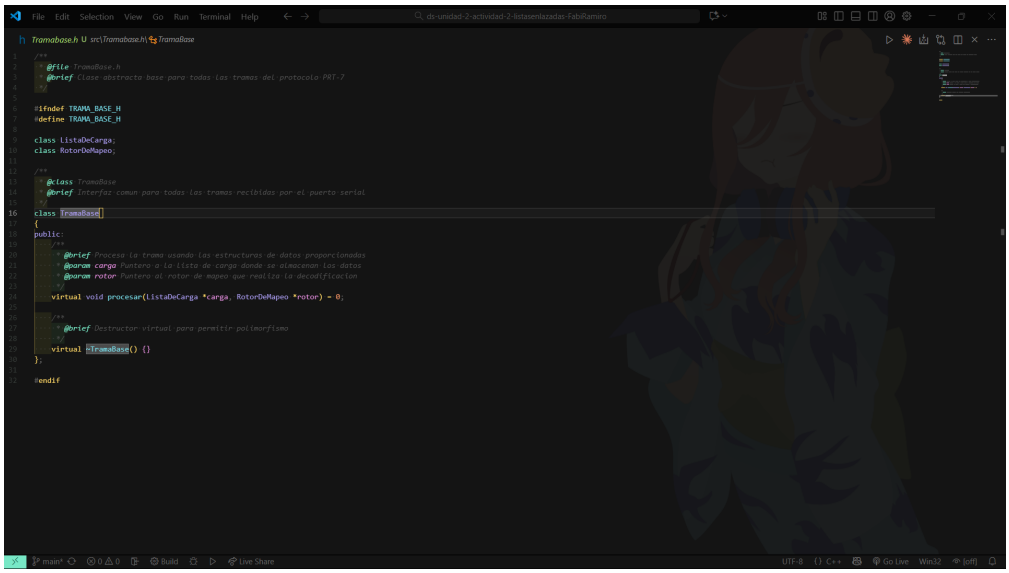


Figura 2: Codigo fuente de TramaBase.h

### 2.1.3. Protocolo PRT-7

El protocolo define dos tipos de tramas con formato de texto plano:

| Tipo | Formato | Descripcion |
|---|---|---|
| LOAD | L,X | Carga el caracter X para decodificacion usando el estado actual del rotor |
| MAP | M,N | Rota el rotor N posiciones. N puede ser positivo (derecha) o negativo (izquierda) |

Cuadro 1: Tipos de tramas del protocolo PRT-7

**Ejemplos de Tramas:**

- `L,H` → Carga el caracter 'H'
- `M,2` → Rota el rotor +2 posiciones (A se mapea a C)
- `L,` → Carga un espacio
- `M,-2` → Rota el rotor -2 posiciones (vuelve al estado original)

## 2.2 Desarrollo

### 2.2.1. Estructura de Datos: RotorDeMapeo

El rotor es una **lista circular doblemente enlazada** que contiene el alfabeto A-Z. Funciona como un disco de cifrado similar al usado en la maquina Enigma.

**Estructura del Nodo:**

```
struct NodoRotor {
    char letra;            // Letra del alfabeto (A-Z)
    NodoRotor* siguiente; // Puntero circular al siguiente
    NodoRotor* previo;    // Puntero circular al anterior
};
```

Listing 3: Nodo del rotor circular

**Funcionamiento del Rotor:**

1. **Inicializacion:** Se crea una lista circular: A→B→C→...→Z→A (cierra el circulo)
2. **Rotacion:** El puntero `cabeza` se mueve N posiciones en la lista circular
3. **Mapeo:** Dado un caracter de entrada, se encuentra su posicion relativa y se retorna el caracter en esa posicion desde la cabeza actual

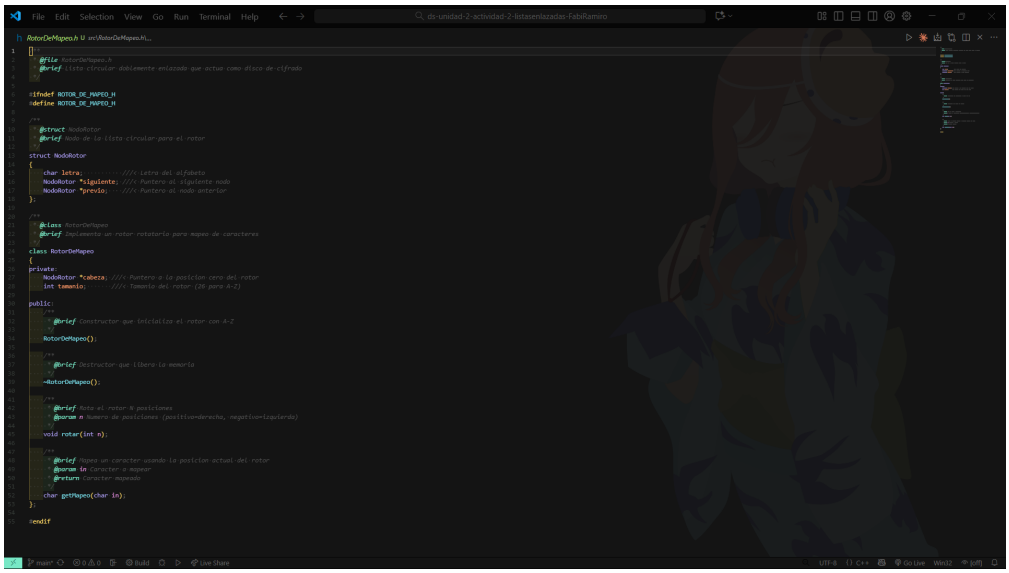


Figura 3: Codigo fuente de RotorDeMapeo.h

**Ejemplo de Rotacion:**

| Estado | Cabeza | A→? | K→? |
|---|---|---|---|
| Inicial | A | A | K |
| Despues de M,2 | C | C | M |
| Despues de M,-2 | A | A | K |

Cuadro 2: Efecto de la rotacion en el mapeo de caracteres

**Implementacion de Rotacion:**

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

    // Normalizar n para evitar rotaciones innecesarias
    n = n % tamanio;

    // Convertir rotaciones negativas a positivas equivalentes
    if (n < 0) {
        n = tamanio + n;
    }

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

Listing 4: Metodo rotar() del RotorDeMapeo

**Implementacion de Mapeo:**

```
char RotorDeMapeo::getMapeo(char in) {
    // Manejar minusculas
    bool eraMinuscula = false;
    if (in >= 'a' && in <= 'z') {
        in = in - 32;
        eraMinuscula = true;
    }

    // Si no es letra, retornar sin cambios
    if (in < 'A' || in > 'Z') return in;

    // Calcular posicion en el alfabeto
    int posicion = in - 'A';

    // Navegar desde la cabeza
    NodoRotor* actual = cabeza;
    for (int i = 0; i < posicion; i++) {
        actual = actual->siguiente;
    }

    char resultado = actual->letra;

    // Restaurar minuscula si era necesario
    if (eraMinuscula) {
        resultado = resultado + 32;
    }

    return resultado;
}
```

Listing 5: Metodo getMapeo() del RotorDeMapeo

### 2.2.2. Estructura de Datos: ListaDeCarga

La lista de carga es una **lista doblemente enlazada** que almacena los caracteres decodificados en el orden en que fueron procesados.

**Estructura del Nodo:**

```
struct NodoCarga {
    char dato;              // Caracter decodificado
    NodoCarga* siguiente;   // Puntero al siguiente nodo
    NodoCarga* previo;      // Puntero al nodo anterior
};
```

Listing 6: Nodo de la lista de carga

**Operaciones Principales:**

```
void ListaDeCarga::insertarAlFinal(char dato) {
    NodoCarga* nuevo = new NodoCarga;
    nuevo->dato = dato;
    nuevo->siguiente = nullptr;
    nuevo->previo = cola;

    if (cola != nullptr) {
        cola->siguiente = nuevo;
    } else {
        // Lista vacia, el nuevo es tambien cabeza
        cabeza = nuevo;
    }

    cola = nuevo;
}
```

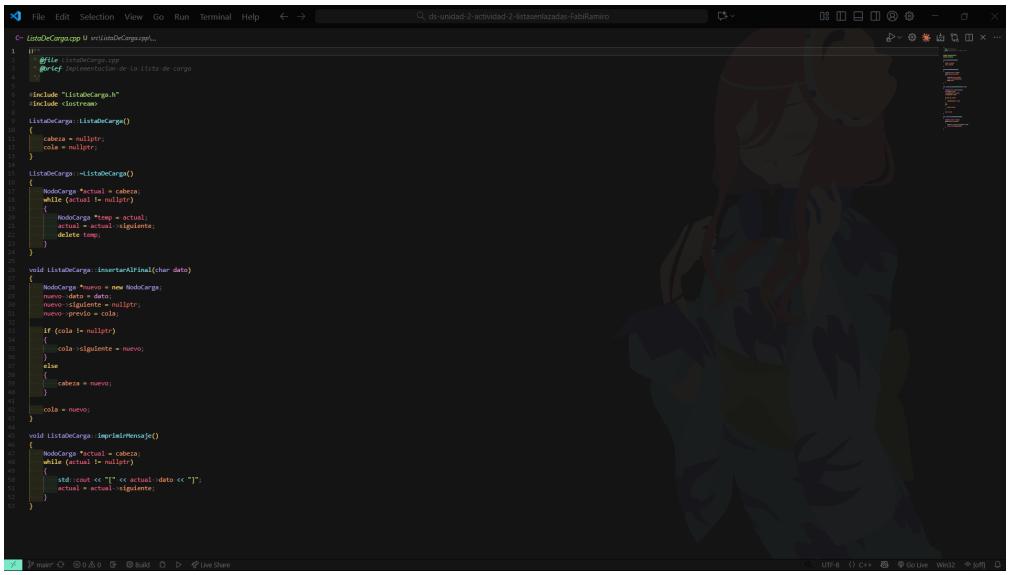
Listing 7: Insercion al final de la lista

```
void ListaDeCarga::imprimirMensaje() {
    NodoCarga* actual = cabeza;
    while (actual != nullptr) {
        std::cout << "[" << actual->dato << "]";
        actual = actual->siguiente;
    }
}
```

Listing 8: Impresion del mensaje almacenado

```
ListaDeCarga::~ListaDeCarga() {
    NodoCarga* actual = cabeza;
    while (actual != nullptr) {
        NodoCarga* temp = actual;
        actual = actual->siguiente;
        delete temp;  // Liberar cada nodo
    }
}
```

Listing 9: Destructor con liberacion de memoria



```cpp
/**
 * @file ListaDeCarga.cpp
 * @brief Implementación de la lista de carga
 */

#include "ListaDeCarga.h"
#include <iostream>

ListaDeCarga::ListaDeCarga()
{
    cabeza = nullptr;
    cola = nullptr;
}

ListaDeCarga::~ListaDeCarga()
{
    NodoCarga *actual = cabeza;
    while (actual != nullptr)
    {
        NodoCarga *temp = actual;
        actual = actual->siguiente;
        delete temp;
    }
}

void ListaDeCarga::insertarAlFinal(char dato)
{
    NodoCarga *nuevo = new NodoCarga;
    nuevo->dato = dato;
    nuevo->siguiente = nullptr;
    nuevo->previo = cola;

    if (cola != nullptr)
    {
        cola->siguiente = nuevo;
    }
    else
    {
        cabeza = nuevo;
    }

    cola = nuevo;
}

void ListaDeCarga::imprimirMensaje()
{
    NodoCarga *actual = cabeza;
    while (actual != nullptr)
    {
        std::cout << "[" << actual->dato << "]";
        actual = actual->siguiente;
    }
}
```

Figura 4: Codigo fuente de ListaDeCarga.cpp

### 2.2.3. Comunicacion Serial

La clase `SerialPort` maneja la comunicacion con el Arduino usando la API de Windows (Win32).

**Configuracion del Puerto COM:**

```
SerialPort::SerialPort(const char* portName) {
    // Abrir puerto COM
    hSerial = CreateFileA(portName,
                          GENERIC_READ | GENERIC_WRITE,
                          0,
                          NULL,
                          OPEN_EXISTING,
                          FILE_ATTRIBUTE_NORMAL,
                          NULL);

    if (hSerial == INVALID_HANDLE_VALUE) {
        conectado = false;
        return;
    }

    // Configurar parametros de comunicacion
    DCB dcbSerialParams = {0};
    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
    dcbSerialParams.BaudRate = CBR_9600;
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.StopBits = ONESTOPBIT;
    dcbSerialParams.Parity = NOPARITY;

    SetCommState(hSerial, &dcbSerialParams);

    // Configurar timeouts
    COMMTIMEOUTS timeouts = {0};
    timeouts.ReadIntervalTimeout = 100;
    timeouts.ReadTotalTimeoutConstant = 500;
    timeouts.ReadTotalTimeoutMultiplier = 100;

    SetCommTimeouts(hSerial, &timeouts);

    // Limpiar buffers
    PurgeComm(hSerial, PURGE_RXCLEAR | PURGE_TXCLEAR);

    conectado = true;
}
```

Listing 10: Apertura y configuracion del puerto serial

**Lectura de Lineas:**

El metodo `leerLinea()` lee caracteres uno por uno hasta encontrar un salto de linea (\n), formando asi una trama completa.

```
bool SerialPort::leerLinea(char* buffer, int bufferSize) {
    if (!conectado) return false;

    DWORD bytesLeidos;
```

```
    int indice = 0;

    // Leer caracter por caracter
    while (indice < bufferSize - 1) {
        char c;
        if (ReadFile(hSerial, &c, 1, &bytesLeidos, NULL)
            && bytesLeidos > 0) {

            if (c == '\r') continue;  // Ignorar retorno de carro

            if (c == '\n') {
                if (indice > 0) {
                    buffer[indice] = '\0';
                    return true;
                }
            } else {
                buffer[indice++] = c;
            }
        }
    }

    buffer[indice] = '\0';
    return (indice > 0);
}
```

Listing 11: Lectura de linea desde puerto serial

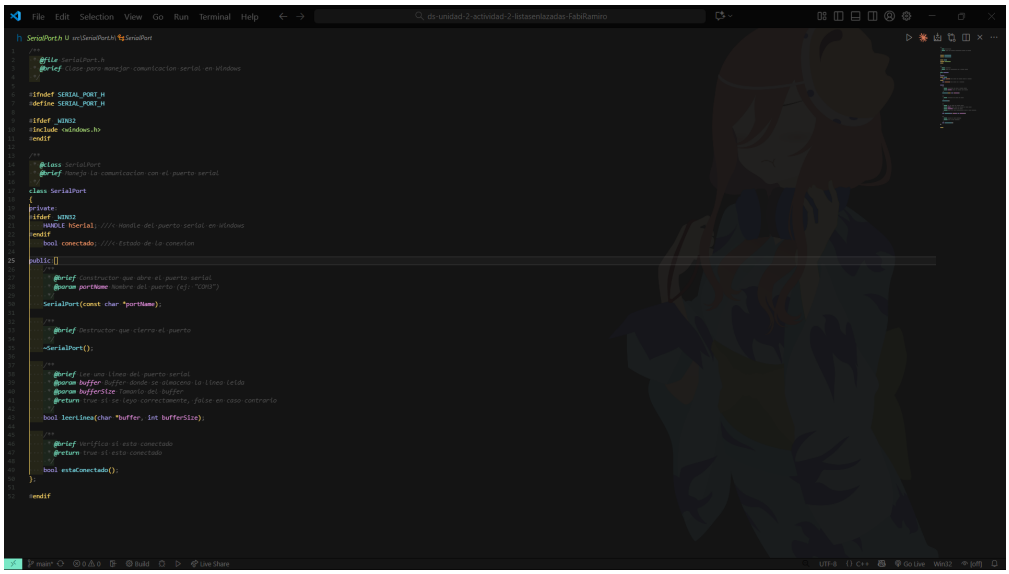


Figura 5: Codigo fuente de SerialPort.h

### 2.2.4. Algoritmo de Decodificacion

El algoritmo principal implementa el bucle de procesamiento de tramas:
**Flujo Principal:**

1. Conectar al puerto serial especificado por el usuario
2. Inicializar rotor con alfabeto A-Z (cabeza en 'A')
3. Inicializar lista de carga vacia
4. Bucle de procesamiento:
   *a*) Leer linea del puerto serial
   *b*) Limpiar espacios y caracteres de control
   *c*) Parsear tipo de trama (L o M)
   *d*) Crear objeto polimórfico correspondiente
   *e*) Ejecutar metodo `procesar()`
   *f*) Liberar memoria del objeto trama
   *g*) Verificar patron de fin
5. Imprimir mensaje decodificado completo
6. Liberar memoria de estructuras de datos
7. Cerrar puerto serial

**Parseo de Tramas:**

```
TramaBase* parsearTrama(const char* linea) {
    if (linea == nullptr || linea[0] == '\0') {
        return nullptr;
    }

    char tipo = linea[0];

    // Buscar la coma separadora
    int i = 0;
    while (linea[i] != ',' && linea[i] != '\0') i++;

    if (linea[i] != ',') return nullptr;

    // Extraer valor despues de la coma
    char* valor = &linea[i + 1];

    // Instanciar la trama correspondiente
    if (tipo == 'L' || tipo == 'l') {
        return new TramaLoad(valor[0]);
    } else if (tipo == 'M' || tipo == 'm') {
        return new TramaMap(atoi(valor));
```

```
    }

    return nullptr;
}
```

Listing 12: Funcion de parseo de tramas

**Deteccion del Patron de Fin:**
El sistema detecta automaticamente el fin del mensaje cuando:

- Se recibe una trama MAP con rotacion negativa (`M,-N`)
- Se han procesado al menos 10 tramas (para evitar falsos positivos)

```
bool esPatronDeFin(const char* linea) {
    if (linea[0] != 'M' && linea[0] != 'm') return false;

    int i = 0;
    while (linea[i] != ',' && linea[i] != '\0') i++;

    if (linea[i] != ',') return false;

    i++;
    while (linea[i] == ' ' || linea[i] == '\t') i++;

    return (linea[i] == '-');
}
```

Listing 13: Deteccion de patron de fin

**Bucle Principal:**

```
while (!mensajeCompleto) {
    if (serial.leerLinea(buffer, 100)) {
        // Limpiar espacios
        // ... codigo de limpieza ...

        // Parsear trama
        TramaBase* trama = parsearTrama(buffer);

        if (trama != nullptr) {
            // Procesar usando polimorfismo
            trama->procesar(&listaCarga, &rotor);
            tramasProcesadas++;

            // Verificar fin
            if (esPatronDeFin(buffer) &&
                tramasProcesadas >= 10) {
                mensajeCompleto = true;
            }

            // CRITICO: Liberar memoria
            delete trama;
        }
    }
```

```
24 }
```

Listing 14: Bucle principal de decodificacion

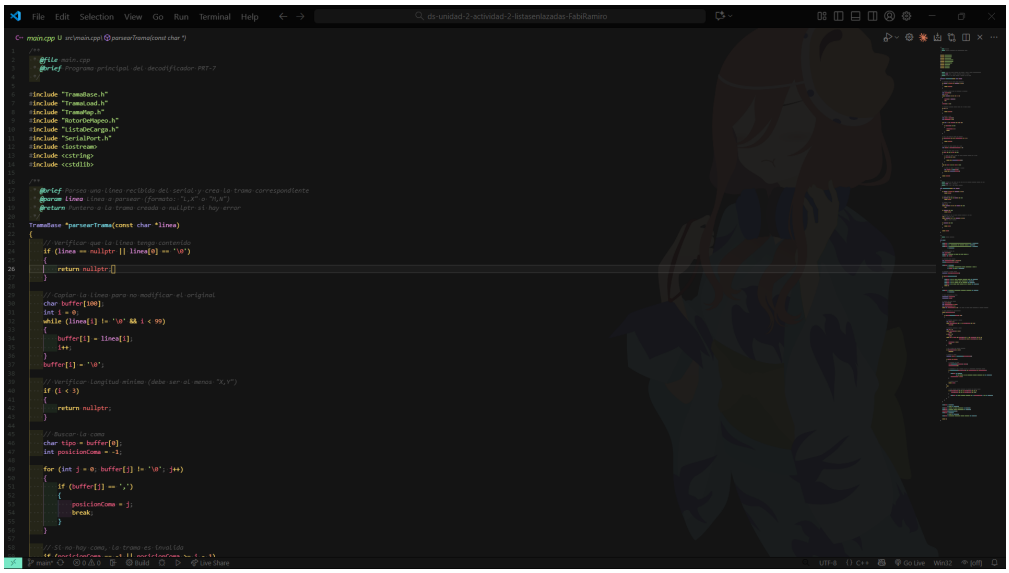


Figura 6: Codigo fuente de main.cpp

## 2.3 Componentes del Sistema

### 2.3.1. Modulo de Tramas

**Archivos:**

- `TramaBase.h` - Clase abstracta base con metodo virtual puro
- `TramaLoad.h/cpp` - Implementacion de tramas de carga
- `TramaMap.h/cpp` - Implementacion de tramas de mapeo

**Responsabilidades:**

- Definir interfaz comun para procesamiento polimórfico
- Implementar logica especifica de cada tipo de trama
- Interactuar con las estructuras de datos (rotor y lista)
- Mostrar progreso en consola

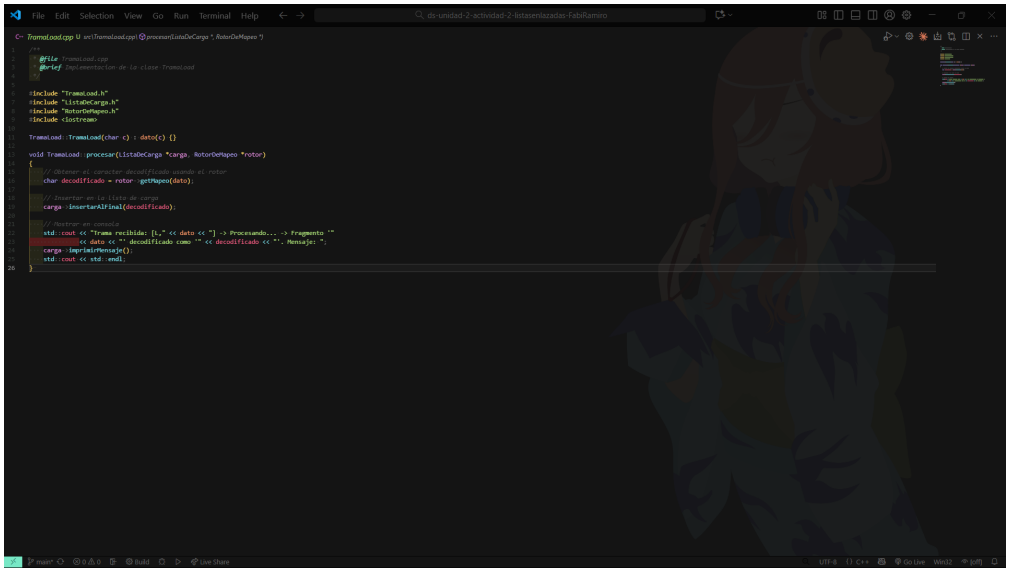


Figura 7: Codigo fuente de TramaLoad.cpp

### 2.3.2. Modulo de Estructuras de Datos

**Archivos:**

- `RotorDeMapeo.h/cpp` - Lista circular doblemente enlazada
- `ListaDeCarga.h/cpp` - Lista doblemente enlazada

**Responsabilidades:**

- Implementar operaciones de listas sin usar STL
- Gestionar memoria dinamica con new/delete
- Proporcionar operaciones eficientes de insercion y rotacion
- Implementar destructores para evitar fugas de memoria

#### 2.3.3. Modulo de Comunicacion

**Archivos:**

- `SerialPort.h/cpp` - Interfaz serial para Windows

**Responsabilidades:**

- Abrir y configurar puerto COM usando Win32 API
- Leer datos en tiempo real del Arduino
- Manejar timeouts y errores de comunicacion
- Limpiar buffers de entrada/salida

### 2.3.4. Programa Principal

**Archivo:**

- `main.cpp` - Punto de entrada del programa

**Responsabilidades:**

- Solicitar puerto COM al usuario
- Coordinar todos los modulos del sistema
- Implementar bucle principal de procesamiento
- Detectar patron de fin automaticamente
- Mostrar resultados y mensajes de estado
- Liberar recursos al finalizar

### 2.3.5. Sistema de Compilacion

El proyecto utiliza **CMake** como sistema de compilacion multiplataforma, permitiendo generar archivos de proyecto para diferentes IDEs y compiladores.

**Archivo CMakeLists.txt:**

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

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED True)

# Buscar Doxygen
find_package(Doxygen)

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

# Ejecutable
add_executable(PRT7Decoder ${SOURCES})

# Target de documentacion
if(DOXYGEN_FOUND)
    add_custom_target(docs
        COMMAND ${DOXYGEN_EXECUTABLE}
                ${CMAKE_SOURCE_DIR}/docs/Doxyfile
        WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}
```

```
        COMMENT "Generando documentacion con Doxygen"
        VERBATIM
    )
endif()
```

Listing 15: Configuracion de CMake

**Comandos de Compilacion en Windows:**

```
# Crear directorio de compilacion
mkdir build
cd build

# Generar archivos de compilacion
cmake .. -G "MinGW Makefiles"

# Compilar el proyecto
mingw32-make

# Ejecutar el programa
./PRT7Decoder.exe

# Generar documentacion (opcional)
cmake --build . --target docs
```

Listing 16: Proceso completo de compilacion

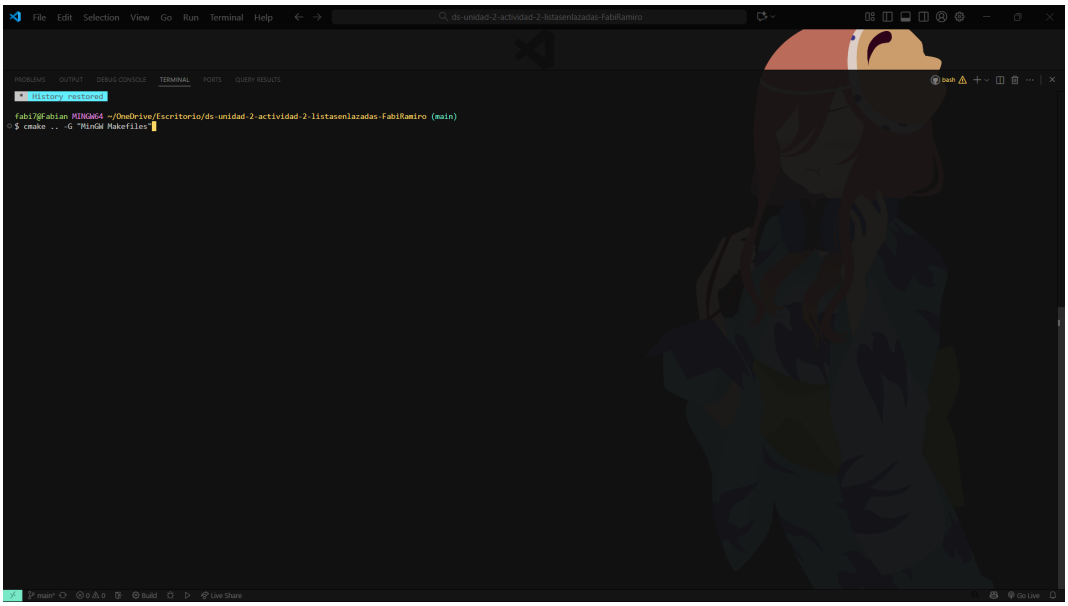



Figura 8: Proceso de compilacion con CMake

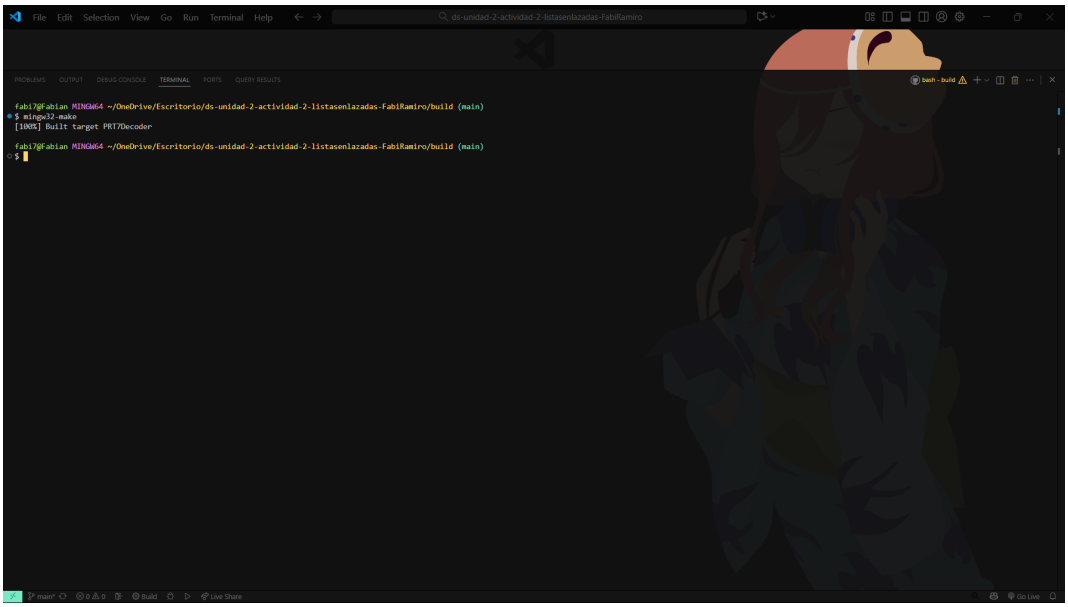



Figura 9: Compilacion con MinGW Make

### 2.3.6. Documentacion con Doxygen

El proyecto incluye documentacion automatica generada con **Doxygen**.
**Caracteristicas de la Documentacion:**

- Pagina principal con descripcion completa del proyecto
- Documentacion de todas las clases, estructuras y funciones
- Indice de archivos y miembros
- Referencias cruzadas entre componentes
- Generacion de HTML navegable

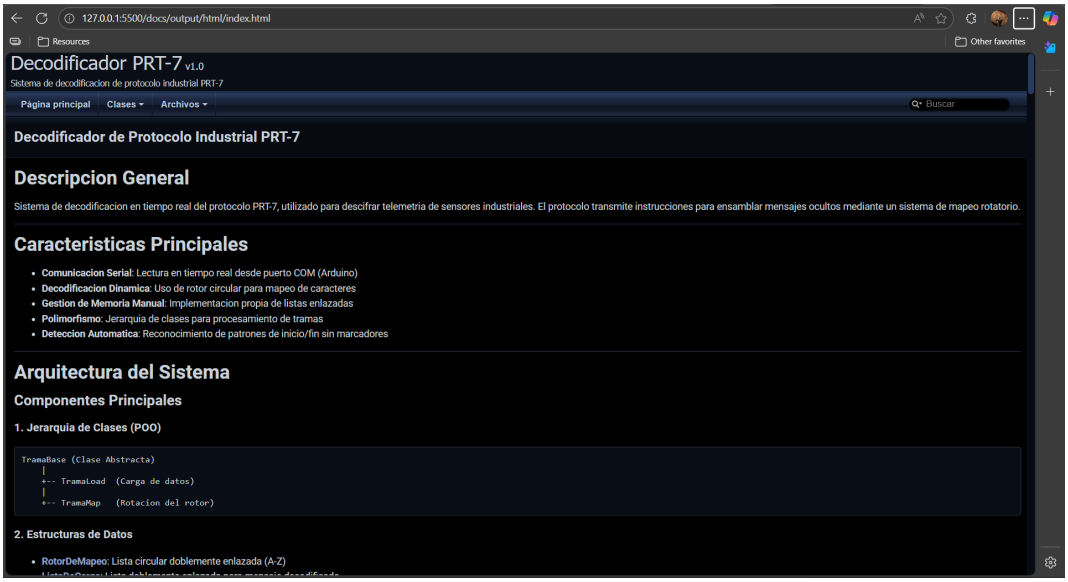



Figura 10: Pagina principal de la documentacion Doxygen

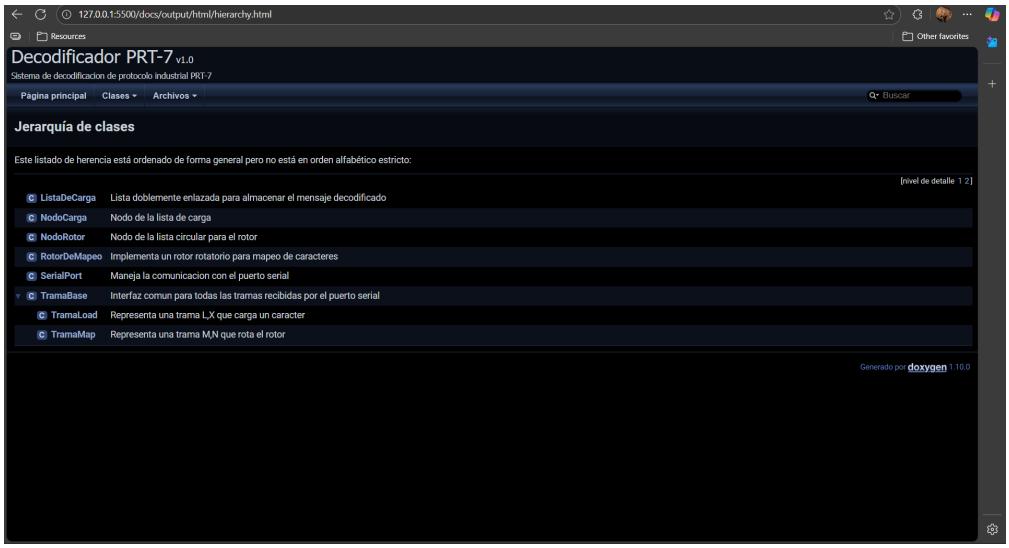



Figura 11: Lista de clases en Doxygen

### 2.3.7. Gestion de Memoria

El proyecto implementa gestion manual de memoria sin garbage collector, siguiendo las mejores practicas de C++.

**Principios Aplicados:**

1. Todo `new` tiene su correspondiente `delete`
2. Destructores virtuales en clases base polimórficas
3. Destructores de listas liberan todos los nodos
4. Romper circularidad antes de eliminar nodos circulares
5. Uso de RAII (Resource Acquisition Is Initialization)

**Ejemplo de Destructor Correcto:**

```
ListaDeCarga::~ListaDeCarga() {
    NodoCarga* actual = cabeza;
    while (actual != nullptr) {
        NodoCarga* temp = actual;
        actual = actual->siguiente;
        delete temp;  // Liberar cada nodo
    }
}
```

Listing 17: Destructor de ListaDeCarga

**Ejemplo de Destructor Circular:**

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

    // CRITICO: Romper el circulo primero
    NodoRotor* ultimo = cabeza->previo;
    ultimo->siguiente = nullptr;

    // Ahora eliminar como lista normal
    NodoRotor* actual = cabeza;
    while (actual != nullptr) {
        NodoRotor* temp = actual;
        actual = actual->siguiente;
        delete temp;
    }
}
```

Listing 18: Destructor de RotorDeMapeo

**Uso de Destructores Virtuales:**

```
// En el main.cpp
TramaBase* trama = new TramaLoad('A');
// ...
delete trama;  // Llama al destructor de TramaLoad gracias
               // al destructor virtual de TramaBase
```

Listing 19: Importancia del destructor virtual

Sin el destructor virtual en `TramaBase`, el `delete` solo llamaria al destructor de la clase base, causando una fuga de memoria.

# 3 Pantallazos de la Implementacion

Esta seccion presenta capturas de pantalla que documentan visualmente el funcionamiento completo del sistema.

## 3.1 Arduino - Codigo Emisor

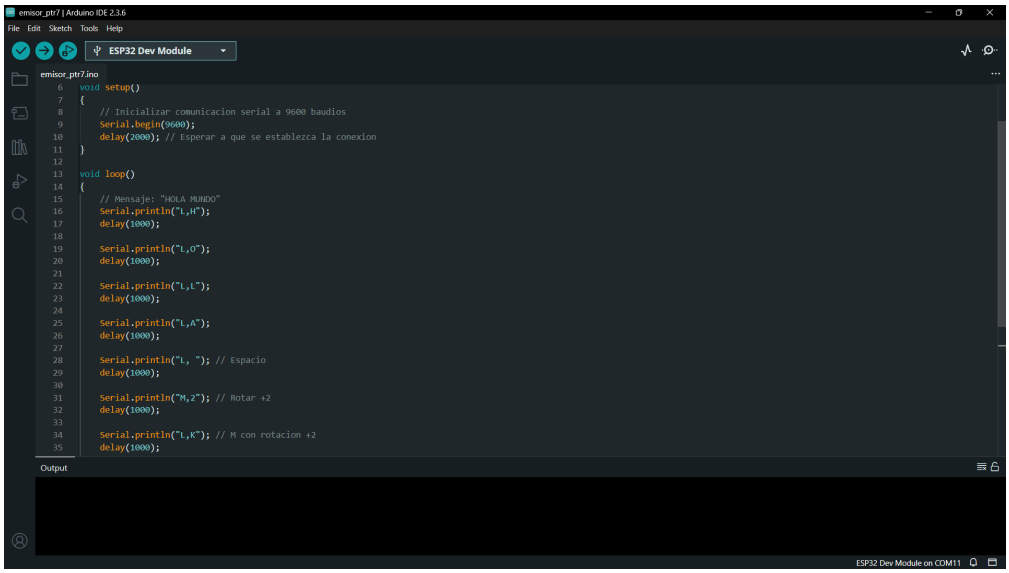



Figura 12: Codigo del emisor PRT-7 en Arduino IDE

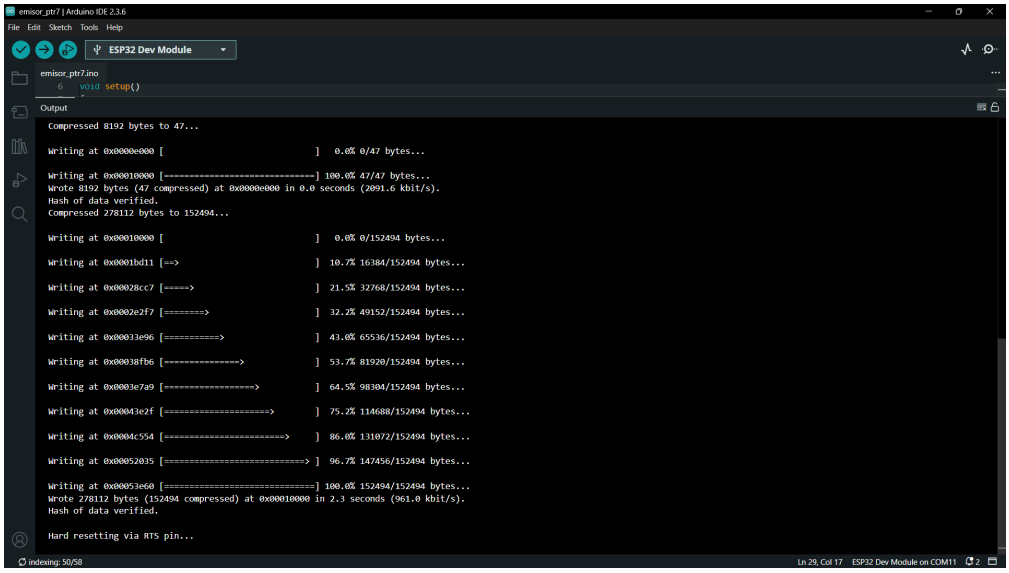



Figura 13: Proceso de subida del codigo al Arduino

## 3.2 Monitor Serial de Arduino

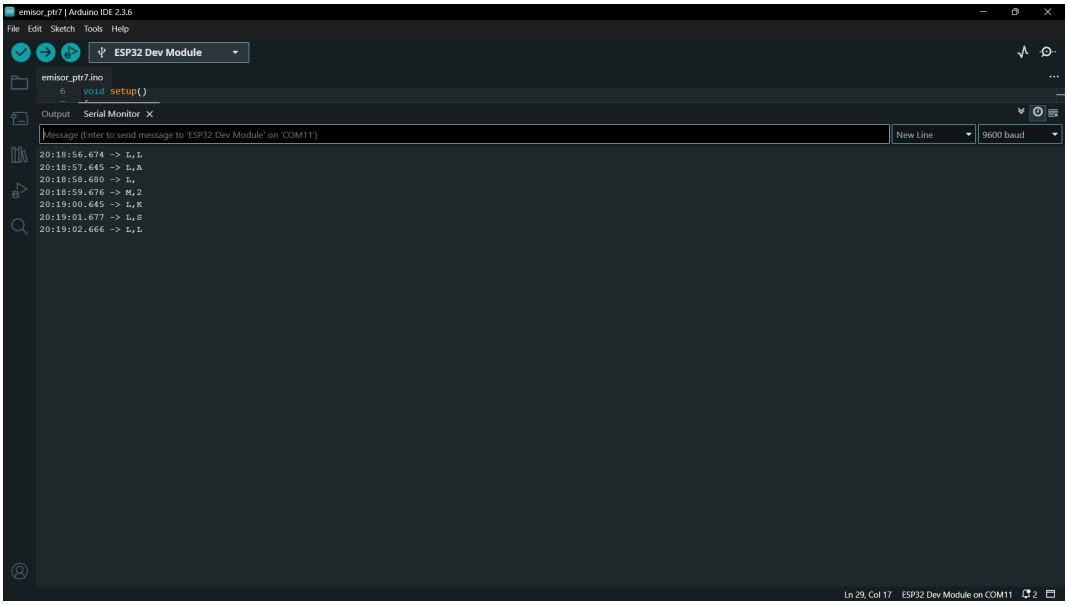



Figura 14: Monitor serial mostrando las tramas transmitidas

## 3.3 Ejecucion del Decodificador

```
$ ./PRT7Decoder.exe

Ingrese el puerto COM (ej: COM3): COM11
```

Figura 15: Pantalla de inicio del decodificador

```
$ ./PRT7Decoder.exe

Ingrese el puerto COM (ej: COM3): COM11

Iniciando Decodificador PRT-7. Conectando a puerto COM11...
Conexion establecida en \\.\COM11
Conexion 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: [L,A] -> Procesando... -> Fragmento 'A' decodificado como 'A'. Mensaje: [H][O][L][A]
Trama invalida recibida: [L,]
Trama recibida: [M,2] -> Procesando... -> ROTANDO ROTOR +2.
```

Figura 16: Procesamiento de tramas en tiempo real

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

---
Flujo de datos terminado.
MENSAJE OCULTO ENSAMBLADO:
[H][O][L][A][M][U][N][D][O]
---
Liberando memoria... Sistema apagado.
```

Figura 17: Mensaje final decodificado

## 3.4 Prueba de Integracion Completa

```
$ ./PRT7Decoder.exe

Ingrese el puerto COM (ej: COM3): COM11

Iniciando Decodificador PRT-7. Conectando a puerto COM11...
Conexion establecida en \\.\COM11
Conexion 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: [L,A] -> Procesando... -> Fragmento 'A' decodificado como 'A'. Mensaje: [H][O][L][A]
Trama invalida recibida: [L,]
Trama recibida: [M,2] -> Procesando... -> ROTANDO ROTOR +2.

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


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

---
Flujo de datos terminado.
MENSAJE OCULTO ENSAMBLADO:
[H][O][L][A][M][U][N][D][O]
---
Liberando memoria... Sistema apagado.
```

Figura 18: Sesion completa de decodificacion de inicio a fin

## 3.5 Documentacion Doxygen Generada

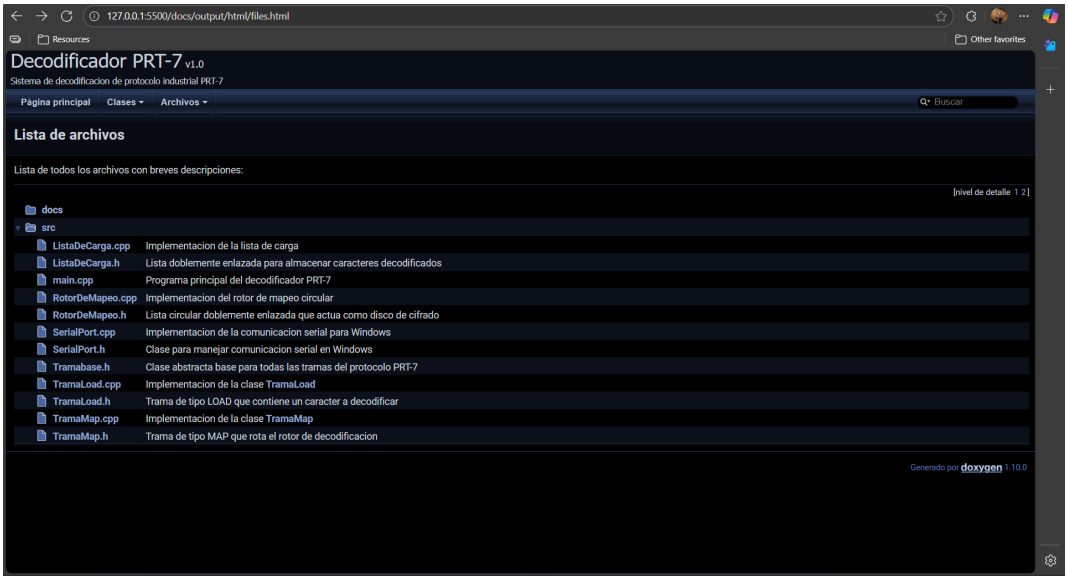



Figura 19: Lista de archivos documentados en Doxygen

## 3.6 Herramientas de Desarrollo

Figura 20: Proyecto abierto en Visual Studio Code

# 4 Conclusiones

## 4.1 Logros Alcanzados

El desarrollo del Decodificador PRT-7 ha cumplido exitosamente todos los objetivos planteados:

1. Se implemento un sistema funcional de decodificacion en tiempo real del protocolo PRT-7
2. Se desarrollo una jerarquia de clases polimorficas (TramaBase, TramaLoad, TramaMap) sin uso de STL
3. Se implementaron manualmente dos estructuras de datos complejas:
   - Lista circular doblemente enlazada (RotorDeMapeo)
   - Lista doblemente enlazada (ListaDeCarga)
4. Se establecio comunicacion serial bidireccional exitosa con dispositivo Arduino
5. Se genero documentacion tecnica completa usando Doxygen
6. Se implemento un sistema de compilacion multiplataforma con CMake
7. El sistema detecta automaticamente el patron de finalizacion sin marcadores explicitos
8. Se logro una gestion correcta de memoria sin fugas detectables

## 4.2 Desafios Enfrentados y Soluciones

Durante el desarrollo se enfrentaron diversos desafios tecnicos:

**Comunicacion Serial en Windows:** La configuracion de la API Win32 para puertos COM resulto compleja. Se soluciono mediante documentacion oficial de Microsoft y ajuste de timeouts.

**Logica del Rotor Circular:** Implementar el mapeo relativo basado en la rotacion actual requirio analisis cuidadoso. Se soluciono calculando posiciones relativas desde la cabeza del rotor.

**Gestion de Memoria en Listas Circulares:** Liberar memoria sin causar fugas en una lista circular fue critico. Se implemento la ruptura del circulo antes de la eliminacion.

**Sincronizacion con Arduino:** El programa debia iniciarse despues del Arduino. Se soluciono haciendo que el Arduino transmita continuamente y el programa capture un ciclo completo.

**Deteccion Automatica de Fin:** Identificar el fin sin marcadores especiales requirio analisis del patron. Se implemento deteccion de rotacion negativa con validacion de tramas minimas.

## 4.3 Aprendizajes Clave

Este proyecto proporciono aprendizajes significativos en multiples areas:
**Conceptuales:**

- Comprension profunda de punteros y referencias en C++
- Dominio de polimorfismo y herencia en diseño orientado a objetos
- Entendimiento de la importancia de destructores virtuales
- Diferencia entre listas circulares y lineales en aplicaciones practicas

**Practicos:**

- Implementacion manual de estructuras de datos sin dependencias externas
- Integracion de software con hardware en tiempo real
- Uso de herramientas profesionales (CMake, Doxygen, Git)
- Debugging de problemas de memoria y comunicacion
- Documentacion de codigo de manera profesional

**Metodologicos:**

- Importancia del diseño antes de la implementacion
- Pruebas incrementales para validacion temprana
- Separacion de responsabilidades en modulos
- Versionamiento de codigo con Git/GitHub

## 4.4 Aplicaciones Reales

El conocimiento adquirido en este proyecto tiene aplicaciones directas en:

- **Sistemas Embebidos:** Comunicacion con dispositivos IoT y sensores
- **Ciberseguridad:** Analisis de protocolos y cifrado de datos
- **Telemetria Industrial:** Decodificacion de datos de SCADA
- **Desarrollo de Firmware:** Programacion de bajo nivel sin librerias externas
- **Sistemas de Tiempo Real:** Procesamiento de streams de datos

## 4.5 Trabajo Futuro

El proyecto puede extenderse en varias direcciones:

1. **Multiples Rotores:** Implementar un sistema tipo Enigma con 3+ rotores en cascada
2. **Cifrado Bidireccional:** Agregar modo de codificacion ademas de decodificacion
3. **Portabilidad:** Portar a Linux/macOS usando termios en lugar de Win32 API
4. **Interfaz Grafica:** Desarrollar GUI con Qt para visualizacion en tiempo real
5. **Optimizaciones:** Implementar rotacion en O(1) usando aritmetica modular
6. **Protocolo Extendido:** Agregar soporte para numeros, simbolos y UTF-8
7. **Persistencia:** Guardar mensajes decodificados en archivos
8. **Red:** Transmision de tramas via TCP/IP en lugar de serial

## 4.6 Reflexion Final

El desarrollo del Decodificador PRT-7 demostro que la implementacion manual de estructuras de datos, aunque mas laboriosa que usar la STL, proporciona un entendimiento profundo de los conceptos fundamentales de programacion.

La integracion de hardware (Arduino) con software (C++), herramientas de compilacion (CMake) y documentacion (Doxygen) simula un entorno de desarrollo profesional real, preparando para proyectos industriales complejos.

El manejo correcto de memoria dinamica y punteros, aunque desafiante, es una habilidad critica para desarrollo de sistemas de alto rendimiento y programacion de bajo nivel.

Este proyecto valida la importancia de las estructuras de datos en la resolucion de problemas reales y complejos.

# 5 Referencias


1. Stroustrup, B. (2013). *The C++ Programming Language (4th Edition)*. Addison-Wesley Professional.

2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms (3rd Edition)*. MIT Press.

3. Weiss, M. A. (2013). *Data Structures and Algorithm Analysis in C++ (4th Edition)*. Pearson.

4. Microsoft Corporation. (2024). *Communications Resources - Win32 apps*. Microsoft Docs. https://docs.microsoft.com/en-us/windows/win32/devio/communications

5. CMake Community. (2024). *CMake Documentation - Version 3.10*. https://cmake.org/documentation

6. van Heesch, D. (2024). *Doxygen Manual - Generating documentation from source code*. https://www.doxygen.nl/manual/

7. Arduino LLC. (2024). *Arduino Language Reference*. https://www.arduino.cc/reference/en/

8. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

9. Meyers, S. (2005). *Effective C++: 55 Specific Ways to Improve Your Programs and Designs (3rd Edition)*. Addison-Wesley Professional.

# A Fragmentos de Codigo Fuente Relevantes

## A.1 TramaBase.h - Clase Abstracta

```
#ifndef TRAMA_BASE_H
#define TRAMA_BASE_H

class ListaDeCarga;
class RotorDeMapeo;

/**
 * Clase base abstracta para todas las tramas PRT-7
 */
class TramaBase {
public:
    /**
     * Metodo virtual puro para procesamiento
     */
    virtual void procesar(ListaDeCarga* carga,
                          RotorDeMapeo* rotor) = 0;

    /**
     * Destructor virtual (CRITICO para polimorfismo)
     */
    virtual ~TramaBase() {}
};

#endif
```

Listing 20: Definicion completa de TramaBase

## A.2 Algoritmo de Parseo

```
TramaBase* parsearTrama(const char* linea) {
    if (linea == nullptr || linea[0] == '\0') {
        return nullptr;
    }

    char buffer[100];
    int i = 0;
    while (linea[i] != '\0' && i < 99) {
        buffer[i] = linea[i];
        i++;
    }
    buffer[i] = '\0';

    if (i < 3) return nullptr;

    char tipo = buffer[0];
    int posicionComa = -1;

```

```
19      for (int j = 0; buffer[j] != '\0'; j++) {
20          if (buffer[j] == ',') {
21              posicionComa = j;
22              break;
23          }
24      }
25
26      if (posicionComa == -1 || posicionComa >= i - 1) {
27          return nullptr;
28      }
29
30      char* valor = &buffer[posicionComa + 1];
31
32      if (tipo == 'L' || tipo == 'l') {
33          return new TramaLoad(valor[0]);
34      } else if (tipo == 'M' || tipo == 'm') {
35          return new TramaMap(atoi(valor));
36      }
37
38      return nullptr;
39  }
```

Listing 21: Parseo de tramas desde texto plano

# B Instrucciones de Instalacion

## B.1 Requisitos del Sistema

**Software:**

- Windows 7 o superior
- MinGW (GCC para Windows)
- CMake 3.10 o superior
- Doxygen (opcional, para documentacion)
- Arduino IDE

**Hardware:**

- Arduino Uno/Nano o compatible
- Cable USB
- Puerto COM disponible

## B.2 Pasos de Instalacion

1. Clonar el repositorio de GitHub
2. Instalar MinGW y agregar al PATH
3. Instalar CMake
4. Compilar el proyecto siguiendo los pasos de la seccion 2.3.5
5. Programar el Arduino con el codigo emisor
6. Ejecutar el decodificador