Reporte Técnico - Decodificador PRT-7

Objetivo del Sistema

El sistema desarrollado tiene como objetivo principal decodificar mensajes transmitidos a través del protocolo PRT-7, el cual no envía datos directamente, sino instrucciones para construir mensajes mediante dos tipos de tramas:

- **Tramas LOAD (L)**: Contienen fragmentos de datos individuales (caracteres)
- **Tramas MAP (M)**: Modifican el estado del sistema de mapeo mediante rotaciones

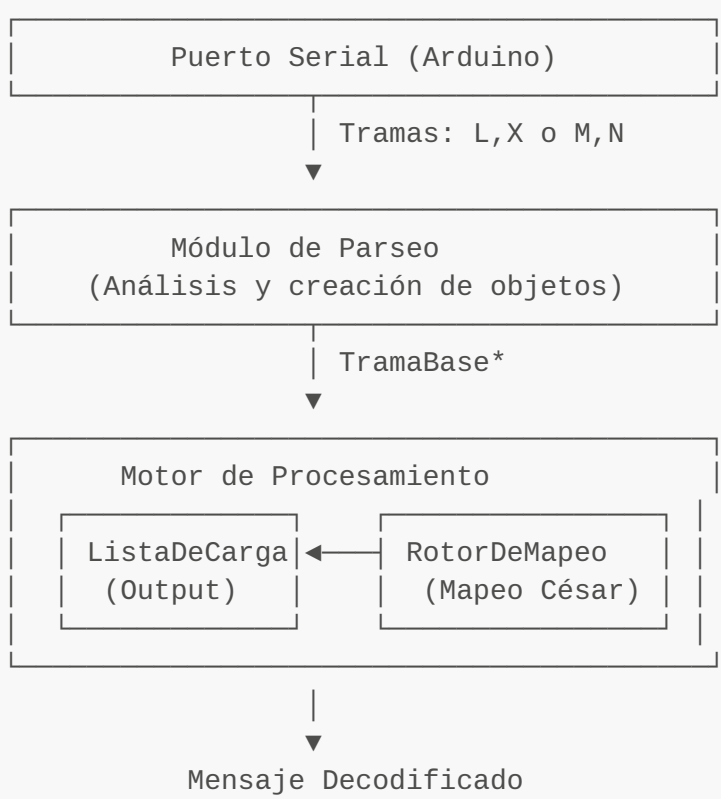
---

Manual Técnico

1. Diseño del Sistema

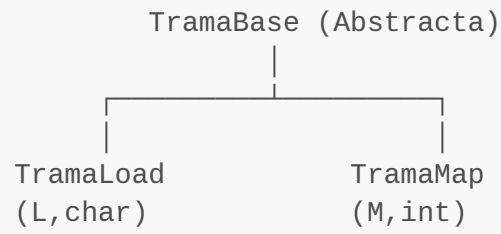
1.1 Arquitectura General

El sistema sigue una arquitectura modular basada en tres componentes principales:



1.2 Jerarquía de Clases (POO)

El diseño utiliza polimorfismo para manejar diferentes tipos de tramas:



TramaBase - Clase base abstracta

- Método virtual puro: `procesar(ListaDeCarga*, RotorDeMapeo*)`
- Destructor virtual para correcta liberación polimórfica

TramaLoad - Trama de carga

- Almacena un carácter
- Decodifica usando el rotor y almacena en la lista

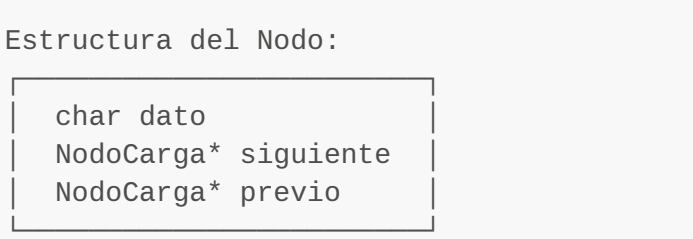
TramaMap - Trama de mapeo

- Almacena un valor de rotación (±N)
- Modifica el estado del rotor

1.3 Estructuras de Datos

Lista Doblemente Enlazada (ListaDeCarga)

Almacena los caracteres decodificados en orden de llegada.



Operaciones principales:

- `insertarAlFinal(char)`: O(1) - mantiene puntero cola
- `imprimirMensaje()`: O(n) - recorrido lineal

Lista Circular Doblemente Enlazada (RotorDeMapeo)

Implementa una rueda de César con 27 elementos (A-Z + espacio).

```
Estado inicial (cabeza apuntando a 'A'):
        ┌─→ [A] ↔ [B] ↔ [C] ↔ ... ↔ [Z] ↔ [ ] ─┐
        └──────────────────────────────────────┘

Después de rotar(2):
        ┌─→ [C] ↔ [D] ↔ [E] ↔ ... ↔ [ ] ↔ [A] ↔ [B] ─┐
        └──────────────────────────────────────────┘
```

Operaciones principales:

- `rotar(int N)`: O(|N|) - mueve puntero cabeza
- `getMapeo(char)`: O(n) - busca y mapea carácter

2. Desarrollo e Implementación

2.1 Flujo de Ejecución Principal

```
Pseudocódigo del proceso principal:

1. Inicializar estructuras:
   - ListaDeCarga (vacía)
   - RotorDeMapeo (A-Z + espacio, cabeza en 'A')

2. Abrir puerto serial (COM3/ttyUSB0)

3. Mientras haya datos:
   a. Leer línea del puerto serial
   b. Parsear línea:
      - Si "L,X" → crear TramaLoad(X)
      - Si "M,N" → crear TramaMap(N)
   c. Ejecutar: trama->procesar(&lista, &rotor)
   d. Liberar: delete trama

4. Mostrar mensaje final
5. Liberar memoria y cerrar puerto
```

2.2 Algoritmo de Decodificación

El algoritmo de mapeo funciona de la siguiente manera:

Caso de TramaLoad:

```
1. Recibir carácter 'X' de la trama
2. Buscar posición de 'X' en el rotor desde cabeza:
   - Posición relativa = índice de 'X' respecto a cabeza
3. Obtener carácter en esa posición relativa desde cabeza
4. Insertar carácter mapeado en ListaDeCarga
```

Ejemplo práctico:

```
Rotor inicial: cabeza → A B C D E F G H...
Recibe: L,G

1. Buscar 'G': está en posición 6 desde cabeza (A)
2. Contar 6 posiciones desde cabeza: A→B→C→D→E→F→G
3. El carácter mapeado es 'G'
4. Insertar 'G' en la lista

Después de M,2:
Rotor rotado: cabeza → C D E F G H I J...
Recibe: L,A

1. Buscar 'A': está en posición -2 (o 25) en el círculo
2. Mapear: 'A' se decodifica como 'C' (la nueva cabeza)
3. Insertar 'C' en la lista
```

2.3 Gestión de Memoria

Principios aplicados:

1. **RAII limitado**: Los destructores liberan recursos automáticamente
2. **Ownership claro**: Cada estructura es dueña de sus nodos
3. **Destructor virtual**: Permite polimorfismo seguro

```
// Patrón de uso correcto:
TramaBase* trama = new TramaLoad('H');  // Crear
trama->procesar(&lista, &rotor);        // Usar
delete trama;                           // Liberar ✓
```

Prevención de fugas:

- Cada `new` tiene su `delete` correspondiente
- Los destructores de listas recorren y liberan todos los nodos
- El destructor virtual en `TramaBase` asegura limpieza correcta

2.4 Comunicación Serial

Windows (Win32 API):

```
- Abrir puerto: CreateFileA("\\\\.\\COM3", ...)
- Configurar: DCB con 9600 baudios, 8N1
- Leer: ReadFile() con timeouts configurados
- Cerrar: CloseHandle()
```

Linux (POSIX termios):

```
- Abrir puerto: open("/dev/ttyUSB0", O_RDONLY)
- Configurar: termios con cfsetispeed(B9600)
- Leer: read() en modo no bloqueante
- Cerrar: close()
```

Modo de prueba: Si el puerto falla, el sistema usa datos predefinidos del README automáticamente.

3. Componentes del Sistema

3.1 Módulo de Clases Base (TramaBase.h)

Propósito: Define la interfaz común para todas las tramas.

Componentes clave:

- Método virtual puro `procesar()`
- Destructor virtual obligatorio
- Forward declarations para evitar dependencias circulares

Responsabilidades:

- Establecer contrato para clases derivadas
- Permitir polimorfismo en tiempo de ejecución

3.2 Módulo de Tramas de Carga (TramaLoad.h/cpp)

Propósito: Representa fragmentos de datos a decodificar.

Atributos:

- `char caracter`: Dato a procesar

Métodos:

- `procesar()`: Mapea el carácter usando el rotor y lo agrega a la lista
- Muestra progreso con formato `[X][Y][Z]`

Casos especiales:

- Maneja "Space" como espacio literal ' '

3.3 Módulo de Tramas de Mapeo (TramaMap.h/cpp)

Propósito: Modifica el estado del sistema de cifrado.

Atributos:

- `int rotacion`: Valor de rotación (puede ser negativo)

Métodos:

- `procesar()`: Rota el rotor N posiciones
- Muestra información de rotación

3.4 Módulo del Rotor (RotorDeMapeo.h/cpp)

Propósito: Implementa la rueda de César dinámica.

Estructura:

- Lista circular de 27 nodos (A-Z + espacio)
- Puntero `cabeza` que marca la posición cero

Métodos críticos:

rotar(int N):

- Complejidad: O(|N|)
- Maneja rotaciones positivas y negativas
- Usa navegación circular con `siguiente`/`previo`

getMapeo(char in):

- Complejidad: O(n) donde n=27
- Busca el carácter en el rotor
- Calcula posición relativa y retorna el mapeo

Invariante: El rotor siempre forma un círculo completo.

3.5 Módulo de Lista de Carga (ListaDeCarga.h/cpp)

Propósito: Almacena el mensaje decodificado en orden.

Estructura:

- Lista doblemente enlazada lineal
- Punteros `cabeza` y `cola` para eficiencia

Métodos:

insertarAlFinal(char):

- Complejidad: O(1)
- Mantiene puntero `cola` actualizado
- Maneja caso de lista vacía

imprimirMensaje():

- Complejidad: O(n)
- Recorre e imprime caracteres secuencialmente

obtenerMensaje():

- Retorna `char*` con el mensaje
- El llamador debe liberar la memoria

3.6 Módulo Principal (main.cpp)

Responsabilidades:

1. Gestión del puerto serial
2. Bucle de lectura de tramas
3. Parseo de cadenas C-style
4. Creación polimórfica de objetos
5. Coordinación del procesamiento

Funciones auxiliares:

abrirPuertoSerial():

- Multiplataforma (Win32/POSIX)
- Configura baudios, paridad, bits de parada
- Retorna handle o descriptor

leerLineaSerial():

- Buffer interno estático
- Maneja caracteres de fin de línea
- Retorna líneas completas

parsearTrama():

- Analiza formato "X,Y"
- Usa `atoi()` para convertir números
- Retorna `TramaBase*` (polimorfismo)

3.7 Sistema de Compilación (CMakeLists.txt)

Características:

- Compatible con CMake 3.10+
- Genera ejecutable multiplataforma
- Integración opcional con Doxygen
- Target `doc` para generar documentación

Configuración:

```
- Estándar: C++11
- Archivos fuente: main, Rotor, Lista, Tramas
- Sin dependencias externas
- Instalación en directorio bin/
```

3.8 Arduino (arduino_prt7.ino)

Propósito: Simula el dispositivo que envía tramas.

Funcionamiento:

- Baudios: 9600
- Envía array predefinido de tramas
- Delay de 1 segundo entre tramas
- Reinicia el ciclo automáticamente

Tramas de prueba:

```
L,H → L,O → L,L → M,2 → L,A → L,Space →
L,W → M,-2 → L,O → L,R → L,L → L,D
```

4. Detalles de Implementación

4.1 Parseo Sin STL

Como no se permite `std::string`, el parseo se realiza con funciones C:

```
// Extraer tipo de trama
char tipo = linea[0];  // 'L' o 'M'

// Verificar formato
if (linea[1] != ',') return nullptr;

// Para MAP: convertir número
if (tipo == 'M') {
    int rotacion = atoi(&linea[2]);  // Maneja ± automáticamente
}

// Para LOAD: extraer carácter
if (tipo == 'L') {
    char c = linea[2];
    // Caso especial "Space"
    if (strcmp(&linea[2], "Space") == 0) c = ' ';
}
```

4.2 Manejo de Rotaciones Negativas

El rotor maneja rotaciones bidireccionales:

```
void RotorDeMapeo::rotar(int N) {
    if (N > 0) {
        // Rotar derecha: A→B→C
```

```
            for (int i = 0; i < N; i++) {
                cabeza = cabeza->siguiente;
            }
        } else {
            // Rotar izquierda: A→Z→Y
            for (int i = 0; i < -N; i++) {
                cabeza = cabeza->previo;
            }
        }
    }
```

4.3 Ejemplo de Ejecución Completo

Entrada: Tramas del README

```
L,H → L,O → L,L → M,2 → L,A → L,Space → L,W → M,-2 → L,O → L,R → L,L → L,D
```

Proceso de decodificación paso a paso:

Trama	Rotor (cabeza)	Acción	Char Decodificado	Mensaje Acumulado
L,H	A	H en pos 7 → H	H	[H]
L,O	A	O en pos 14 → O	O	[H][O]
L,L	A	L en pos 11 → L	L	[H][O][L]
M,2	**A→C**	**Rotar +2**	-	[H][O][L]
L,A	C	A en pos -2 → C	C	[H][O][L][C]
L,Space	C	Space en pos 24 → Space	(espacio)	[H][O][L][C][ ]
L,W	C	W en pos 20 → Y	Y	[H][O][L][C][ ][Y]
M,-2	**C→A**	**Rotar -2**	-	[H][O][L][C][ ][Y]
L,O	A	O en pos 14 → O	O	[H][O][L][C][ ][Y][O]
L,R	A	R en pos 17 → R	R	[H][O][L][C][ ][Y][O][R]
L,L	A	L en pos 11 → L	L	[H][O][L][C][ ][Y][O][R][L]
L,D	A	D en pos 3 → D	D	[H][O][L][C][ ][Y][O][R][L][D]

Salida Final: `HOLC YORLD`