# Laboratorio Calificado 01

## 2025

# ÍNDICE

1 Instrucciones ..... 3
2 Contexto ..... 3
3 Requisitos del Problema ..... 3
3.1 Tablero: ..... 3
3.2 Funcionalidad del Robot: ..... 3
3.3 Interacción con el Usuario: ..... 4
3.4 Conceptos de Programación Orientada a Objetos: ..... 4
3.5 Movimiento y Visualización: ..... 4
4 Tareas para Resolver ..... 4
4.1 Abstracción e Identificación: ..... 4
4.2 Modelado: ..... 4
4.3 Programación: ..... 4
5 Entregable ..... 5
6 Código Base para Comenzar ..... 5
7 Notas ..... 6
8 Caso de Prueba: Inspección de la Estación 7 ..... 6
8.1 Descripción ..... 6
9 Rúbrica de Evaluación ..... 7

# Laboratorio Calificado 01
# Robot Inspector de Fábrica en un Tablero

## 1 Instrucciones

Se ha creado la tarea “Laboratorio Calificado 1” en la sección Laboratorio Calificado 1 de la página del curso-horario en PAIDEIA. Cada alumno deberá entregar un archivo comprimido (formato ZIP) con todos los archivos del desarrollo realizado en el laboratorio. El nombre del archivo comprimido deberá tener el formato:
L1_<Código del alumno de 8 dígitos>.<Extensión del archivo>
Como ejemplo, el nombre del archivo de alumno 20236969 del horario 06M1 sería “L1_20236969.zip”. Es COMPLETA RESPONSABILIDAD DE CADA ALUMNO el colocar nombres correctos a sus archivos y evitar así confusiones al momento de la calificación.

## 2 Contexto

En una fábrica de ensamblaje de componentes mecatrónicos, se utiliza un robot para inspeccionar estaciones de trabajo distribuidas en un área de producción representada por un tablero de 5 filas por 10 columnas (5x10 metros cuadrados). Cada celda del tablero equivale a un metro cuadrado y puede estar vacía o contener una estación de trabajo. Hay exactamente 10 estaciones numeradas del 1 al 10, distribuidas de manera que siempre estén rodeadas por celdas vacías, permitiendo al robot moverse entre ellas. El robot comienza en la esquina superior izquierda del tablero (posición inicial) y debe desplazarse a posiciones adyacentes a las estaciones para realizar inspecciones. El sistema debe ser interactivo, permitiendo al usuario ingresar comandos para controlar al robot hasta que decida finalizar la operación.

## 3 Requisitos del Problema

Su tarea es diseñar y programar un sistema en C++/CLI utilizando programación orientada a objetos para modelar este escenario:

### 3.1 Tablero:

- Representa el área de producción como un tablero de 5x10 celdas.
- Incluye 10 estaciones de trabajo numeradas del 1 al 10, distribuidas de izquierda a derecha y de arriba hacia abajo, rodeadas de celdas vacías (representadas por 0). Ejemplo de distribución posible:

```
0  1  0  2  0  0  0  0  0  0
0  0  0  0  0  3  0  4  0  0
0  5  0  6  0  0  0  0  0  0
0  0  0  0  0  7  0  8  0  0
0  9  0 10  0  0  0  0  0  0
```

- El robot debe poder moverse por las celdas vacías para llegar a las estaciones.

### 3.2 Funcionalidad del Robot:

- El robot debe ser capaz de:
  - Inspeccionar una estación específica (indicada por su número) moviéndose a una celda vacía adyacente (arriba, abajo, izquierda o derecha).
  - Detenerse cuando se le indique.
  - Reportar su estado, incluyendo información relevante como el número de inspecciones realizadas y su posición actual.
- Debe existir una versión avanzada del robot que:

- Pueda recargar su batería (o un recurso similar).
- Proporcione un reporte más detallado que el robot básico.

### 3.3 Interacción con el Usuario:

- El programa principal debe usar un bucle para solicitar comandos al usuario hasta que se ingrese "Q" para salir.
- Los comandos posibles incluyen:
  - Inspeccionar una estación (solicitando el número de estación, 1-10).
  - Detener al robot.
  - Recargar (solo para la versión avanzada).
    - Reportar el estado del robot.
  - Salir del programa con "Q".
- Usa estructuras condicionales (`if-else`) para procesar los comandos, ya que C++/CLI no soporta `switch` con cadenas directamente.

### 3.4 Conceptos de Programación Orientada a Objetos:

- Usa **variables de instancia** para almacenar información específica del robot (ej. posición, conteo de inspecciones).
- Usa dos **variables globales**, una para llevar un conteo total de inspecciones realizadas en la fábrica y la otra para definir .
- Usa un **diccionario** para almacenar información dinámica, como el estado de las estaciones (ej. "funcional" o "falla").
- Aplica **herencia** para modelar una relación entre un robot básico y una versión avanzada con funcionalidades adicionales.
- Implementa **polimorfismo** para que el reporte del estado varíe entre el robot básico y el avanzado.

### 3.5 Movimiento y Visualización:

- El robot debe moverse paso a paso hacia una celda vacía adyacente a la estación solicitada, verificando que cada paso sea a una celda vacía (0).
- Si no hay camino disponible, informa al usuario.
- Muestra el tablero después de cada movimiento, marcando la posición del robot con "R".

## 4 Tareas para Resolver

### 4.1 Abstracción e Identificación:

- Analiza el contexto y determina qué entidades del problema pueden modelarse como objetos. Considera las acciones que realiza el robot, la información que necesita almacenar y las diferencias entre un robot básico y uno avanzado.
- Identifica las relaciones entre estos objetos (ej. ¿uno puede ser una versión extendida del otro?).

### 4.2 Modelado:

- Diseña las clases necesarias, definiendo:
  - Los atributos que cada clase debe tener (ej. posición, estado, etc.).
  - Los métodos que cada clase debe incluir para cumplir con las funcionalidades descritas (ej. inspeccionar, moverse, reportar).
- Asegúrate de que las clases reflejen herencia y polimorfismo.

### 4.3 Programación:

- Declara el tablero como un arreglo bidimensional en C++/CLI.

- Implementa las clases con sus atributos y métodos, incluyendo la lógica de movimiento y la interacción con el usuario.
- Programa el bucle principal para procesar comandos y mostrar el tablero.

## 5 Entregable

Escribe un programa completo en C++/CLI que:

- Modele el sistema descrito usando programación orientada a objetos.
- Permita al usuario interactuar con el robot mediante comandos.
- Muestre el tablero con la posición del robot tras cada movimiento.
- Implemente todos los conceptos solicitados (variables de instancia/globales, diccionarios, herencia, polimorfismo).

.

## 6 Código Base para Comenzar

**C++/CLI**

```
#include "pch.h"

using namespace System;
using namespace System::Collections::Generic;

// Clase para manejar datos estáticos de la fábrica
ref class FactoryStats {
public:
    static int totalInspections;  // Contador estático de inspecciones totales
};

// Clase para manejar el diseño estático del tablero
ref class FactoryLayout {
public:
    static array<int, 2>^ board = {
        {0, 1, 0, 2, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 3, 0, 4, 0, 0},
        {0, 5, 0, 6, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 7, 0, 8, 0, 0},
        {0, 9, 0, 10, 0, 0, 0, 0, 0, 0}
    };
};

// Define tus clases aquí

int main() {
    // Crea el robot aquí (puedes elegir el nombre)
    String^ command = "";

    // Muestra el tablero inicial (implementa esta funcionalidad en tu clase)

    while (command != "Q") {
        Console::Write("Ingrese comando (inspeccionar, detener, recargar,
reportar, Q para salir): ");
        command = Console::ReadLine();
        // Procesa los comandos aquí usando if-else
    }

    Console::WriteLine("Saliendo... Inspecciones totales realizadas: {0}",
FactoryStats::totalInspections);
    return 0;
}
```

# 7 Notas

- Piensa en cómo el robot básico puede ser extendido para agregar funcionalidades avanzadas.
- Diseña el movimiento del robot como un proceso paso a paso, asegurándote de que solo se mueva por celdas vacías.

# 8 Caso de Prueba: Inspección de la Estación 7

## 8.1 Descripción

Este caso de prueba verifica que el robot inspector pueda moverse desde su posición inicial hasta una celda vacía adyacente a la estación 7 y realizar una inspección, mostrando el tablero en cada paso del movimiento.

- **Condición Inicial:**
  - El robot comienza en la posición inicial (0,0).
  - El tablero está configurado como se muestra en el código base:

```
0  1  0  2  0  0  0  0  0  0
0  0  0  0  0  3  0  4  0  0
0  5  0  6  0  0  0  0  0  0
0  0  0  0  0  7  0  8  0  0
0  9  0 10  0  0  0  0  0  0
```

  - La estación 7 está ubicada en la posición (3,5).
- **Entrada del Usuario:**
  - Comando: inspeccionar
  - Número de estación: 7
- **Comportamiento Esperado:**
  - El programa debe:
    1. Identificar la posición de la estación 7 en (3,5).
    2. Encontrar una celda vacía adyacente a (3,5), como (3,4).
    3. Mover al robot desde (0,0) hasta (3,4) a través de celdas vacías, mostrando el tablero después de cada paso.
    4. Informar que la inspección se realizó en la estación 7.
- **Salida Esperada:**
  - El robot se mueve paso a paso por un camino válido, como: (0,0) → (1,0) → (2,0) → (3,0) → (3,1) → (3,2) → (3,3) → (3,4).
  - Ejemplo de tableros mostrados (solo algunos pasos para ilustrar):

```
Tablero:
R  1  0  2  0  0  0  0  0  0
0  0  0  0  0  3  0  4  0  0
0  5  0  6  0  0  0  0  0  0
0  0  0  0  0  7  0  8  0  0
0  9  0 10  0  0  0  0  0  0

Tablero:
0  1  0  2  0  0  0  0  0  0
0  0  0  0  0  3  0  4  0  0
0  5  0  6  0  0  0  0  0  0
R  0  0  0  0  7  0  8  0  0
0  9  0 10  0  0  0  0  0  0

Tablero:
0  1  0  2  0  0  0  0  0  0
0  0  0  0  0  3  0  4  0  0
0  5  0  6  0  0  0  0  0  0
0  0  0  0  R  7  0  8  0  0
0  9  0 10  0  0  0  0  0  0
```

- Mensaje final: Un texto que indique que el robot ha inspeccionado la estación 7, por ejemplo:
  `InspectorBot inspeccionando estación 7. Estado: funcional`
- El conteo total de inspecciones (`FactoryStats::totalInspections`) debe incrementarse en 1.

- **Notas para el Estudiante:**
  - El camino exacto puede variar dependiendo de cómo implementes el movimiento, pero debe usar solo celdas vacías (0) y llegar a una celda adyacente a la estación 7 (como (3,4), (3,6), (2,5) o (4,5)).
  - Asegúrate de que el tablero se muestre después de cada paso del movimiento, con "R" indicando la posición del robot.

# 9 Rúbrica de Evaluación

**Puntaje Máximo: 20 puntos**

| Criterio | Excelente | Bueno | Regular | Deficiente | No Cumple |
|---|---|---|---|---|---|
| **Abstracción, Identificación de Objetos y Modelado de Clases (2 puntos)** | Identifica correctamente los objetos clave (robot básico y avanzado) y sus relaciones, con una abstracción clara y lógica basada en el contexto. | Identifica los objetos principales con alguna relación, pero la abstracción tiene pequeños errores o falta detalle. | Identifica algunos objetos, pero la abstracción es incompleta o confusa. | Identifica objetos de forma mínima, con abstracción poco clara o irrelevante. | No realiza abstracción ni identifica objetos. |
| **Modelado de Clases (4 puntos)** | Diseña clases completas con atributos y métodos bien definidos, usando encapsulamiento adecuadamente (ej. privados con getters). | Diseña clases con la mayoría de atributos y métodos necesarios, pero con encapsulamiento parcial o pequeños errores. | Diseña clases con atributos y métodos básicos, pero omite encapsulamiento o tiene errores significativos. | Diseña clases incompletas con pocos atributos/métodos o errores graves. | No modela clases o lo hace incorrectamente. |
| **Aplicación de Herencia y Polimorfismo (3 puntos)** | Implementa herencia entre un robot básico y avanzado, con polimorfismo claro (método sobrescrito funcional) y uso correcto de virtual/override. | Implementa herencia y polimorfismo, pero con errores menores en la sintaxis o funcionalidad limitada. | Implementa herencia o polimorfismo parcialmente, con errores que afectan la funcionalidad. | Intenta herencia o polimorfismo, pero con errores graves o sin funcionalidad. | No aplica herencia ni polimorfismo. |
| **Uso de Variables y Diccionarios (3 puntos)** | Usa variables de instancia y globales correctamente (ej. conteo de inspecciones), e implementa un diccionario funcional para el estado de estaciones. | Usa variables y diccionario con funcionalidad básica, pero con errores menores en su manejo o propósito. | Usa variables o diccionario de forma limitada, con errores que afectan el programa. | Usa variables o diccionario de manera incorrecta o incompleta. | No usa variables ni diccionarios adecuadamente. |
| **Funcionalidad y Programación (8 puntos)** | El programa cumple todos los requisitos: mueve al robot, procesa comandos con if-else, muestra el tablero y funciona sin errores. | El programa cumple la mayoría de los requisitos, pero tiene errores menores en movimiento o comandos. | El programa funciona parcialmente (ej. mueve al robot pero falla en algunos comandos o visualización). | El programa tiene funcionalidad mínima con errores graves que impiden su ejecución correcta. | El programa no funciona o está incompleto. |