Trabajo Practico 2

I102 – Paradigmas de Programación – G:2

Fecha límite de entrega: 19/06/2025 – 22hs

La entrega se realizará mediante un archivo comprimido ZIP que incluirá un informe y el código en C++ (.CPP y .HPP o .H). El informe, en formato PDF, debe incluir su nombre, explicar la solución de los problemas, los Warnings indicados por el compilador e incluir los comandos necesarios para compilar el código. En los archivos de C++, se deberán detallar todos los comentarios necesarios para seguir fácilmente la implementación provista. Las penalidades por no poder compilar el código y las advertencias del compilador encontrados con -Wall, -Wpedantic y -Wextra, no explicadas en el informe implicarán las penalidades mencionadas en el programa de la materia (50% por no compilar y entre 5-10% por Warning). El archivo de formato ZIP debe poseer el siguiente nombre TP2_NombreCompleto.zip. Recuerde que el trabajo es en grupos de dos personas (salvo por un único grupo de 3 personas).

1. Pokedex:

Se desea construir un Pokedex digital en C++. Cada tipo de Pokemon tiene una información específica y única. Para este ejercicio es necesario definir tres clases:

- Pokemon: del que se quiere encontrar información
- PokemonInfo: posee la información del pokemon
- Pokedex: actúa como contenedor principal para la información

Cada instancia de las clases debe tener:

Pokemon:

- nombre: el nombre del pokemon (por ejemplo, “Charmander”).
- experiencia: un entero positivo que representa su experiencia actual.

Estos atributos deberán poder ser obtenidos desde esta clase para su impresión por pantalla.

PokemonInfo:

- tipo: un tipo std::string que indica el tipo de Pokemon (agua, fuego, planta, eléctrico, hielo, lucha, veneno o tierra).
- descripcion: una breve descripción del Pokemon.
- ataquesDisponiblesPorNivel: use el contenedor que crea conveniente (esto será evaluado, justifíquelo en el informe) para almacenar los ataques por nombre y puntaje de daño que hacen. Tenga en cuenta que los ataques son enteros positivos.

- experienciaProximoNivel: use el contenedor que crea conveniente (esto será evaluado, justifíquelo en el informe) para almacenar cuanta experiencia (un valor entero positivo) se requiere para alcanzar cada uno de los 3 niveles posibles.

Estos atributos deberán poder ser obtenidos desde esta clase para su impresión por pantalla.

Pokedex:

- Esta clase almacenará la información utilizando un contenedor del tipo std::unordered_map donde la clave o key será una instancia de la clase Pokemon y el valor o value será una instancia de la clase PokemonInfo.
- Implemente los métodos que crea necesarios en la clase Pokemon para poder utilizar sus objetos como key del contenedor std::unordered_map.
- Implemente el hash para el std:: unordered_map como un functor utilizando sólo el atributo nombre de la clase Pokemon.
- Implemente un método denominado “mostrar” que le permita imprimir por pantalla la información del objeto Pokemon pasado como parámetro. Si el Pokemon no puede encontrarse, se deberá imprimir el mensaje “¡Pokemon desconocido!”.

Se pide:

a. La implementación del sistema Pokedex como fue descrito y cumpliendo con los puntos anteriores.

b. Crear tres Pokemons distintos para cargar en el Pokedex e imprimirlo su información por pantalla utilizando un único método denominado “mostrarTodos”. Utilice la siguiente información:

Caso 1: Pokemon("Squirtle", 100)
PokemonInfo("Agua", "Una tortuga pequeña que lanza chorros de agua.",
{{"Pistola Agua", 4}, {"Hidrobomba", 6}, {"Danza Lluvia", 5}},
{0, 400, 1000}));

Caso 2: Pokemon("Bulbasaur", 270),
PokemonInfo("Planta", "Tiene una semilla en su lomo que crece con el tiempo",
{{"Latigazo", 4}, {"Hoja Afilada", 6}, {"Rayo Solar", 5}},
{0, 300, 1100}));

Caso 3: Pokemon("Charmander", 633),
PokemonInfo("Fuego", "Una lagartija con una llama en su cola",
{{"Ascuas", 4}, {"Lanzallamas", 6}, {"Giro Fuego", 5}},
{0, 250, 1300}));

Estos pokemones deben ser reconocidos por el Pokedex simplemente utilizando una instancia de la clase Pokemon. La idea es reconocer su tipo y mostrar por pantalla su información.

c. Cree otro método para el Pokedex, llamado “mostrar” que le permita buscar un pokemon usando un objeto Pokemon. Verifique su correcto funcionamiento con una búsqueda que devuelva un resultado exitoso y otra que indique que el pokemon no se encontró:

   Devuelve Resultado con: Pokemon("Squirtle", 870)
   No Devuelve Resultado con: Pokemon("Pikachu", 390)

d. **Opcional (10 puntos):** Permita que el Pokedex guarde en un archivo la información de los pokemons agregados.

   - El nombre del archivo debe ser pasado al constructor sobrecargado.
   - El Pokedex debe cargar automáticamente la información al ser creado.
   - Toda nueva información agregada debe actualizar el archivo.

2. Control de Aeronave en Hangar Automatizado:

Considere un grupo de cinco drones cuadricópteros ubicados en círculo dentro de un hangar. Cada dron está listo para despegar, pero para hacerlo debe asegurarse de que las dos zonas adyacentes (a su izquierda y a su derecha) estén libres de interferencia. Esto se debe a que la turbulencia generada por drones cercanos puede desestabilizar el vuelo durante el despegue.

Por esta razón, cada dron necesita adquirir el control exclusivo de sus dos zonas laterales antes de iniciar el despegue. Las zonas de interferencia están ubicadas entre cada par de drones vecinos, ver Figura 1. En total hay cinco zonas, una entre cada par de drones consecutivos.

Cuando el dron inicia su funcionamiento realiza las siguientes tareas:

- Se prepara para el despegue verificando si puede adquirir las zonas laterales. Las zonas laterales sólo podrán ser adquiridas si están libres, es decir, si no las tomó ningún otro dron vecino o si estos están a una altura de 10 metros. De no poder tomar las zonas laterales, el dron quedará a la espera de que ambas queden libres.
- Luego, ocupa las dos zonas laterales adyacentes.
- Inicia el despegue y luego de 5 segundos alcanza la altura segura de 10 metros.
- Libera ambas zonas luego de alcanzar los 10 metros de altura.

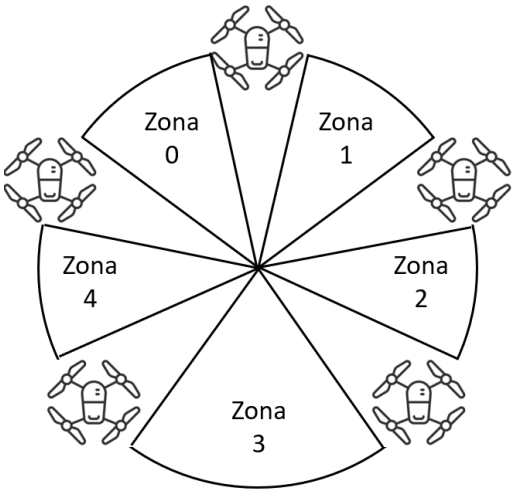

Figura 1: Distribución de los drones y zonas de interferencia en el problema 2.

Para este ejercicio se pide:

1. Escriba un programa en C++ que utilice una clase y programación multihilo, que simule este escenario sólo utilizando std::mutex y std::lock (este puede ser un multilock). También debe minimizar el tiempo total requerido (esto se logra mediante programación multihilo sin utilizar tiempos de espera para sincronizar los eventos, sólo debe existir una espera de 5 segundos en el código) para que todos los drones completen su despegue y alcancen los 10 metros de altura.
2. Escriba un programa para verificar su funcionamiento. El mismo deberá de proveer una salida por pantalla como la siguiente:

```
Dron 0 esperando para despegar...
Dron 0 despegando...
Dron 1 esperando para despegar...
Dron 2 esperando para despegar...
Dron 2 despegando...
Dron 3 esperando para despegar...
Dron 4 esperando para despegar...
Dron 0 alcanzó altura de 10m
Dron 4 despegando...
Dron 2 alcanzó altura de 10m
Dron 1 despegando...
Dron 4 alcanzó altura de 10m
Dron 3 despegando...
Dron 1 alcanzó altura de 10m
Dron 3 alcanzó altura de 10m
```

Note que no se observan impresiones por pantalla donde los mensajes estén mezclados. En la resolución de su ejercicio, no debe observarse que los mensajes por pantalla se superpongan en ningún momento.

3. Sistema de Monitoreo y Procesamiento de Robots Autónomos:

En una planta industrial automatizada, varios sensores realizan tareas de inspección periódica y generan reportes. Estos reportes representan tareas que deben ser procesadas por un conjunto de robots autónomos encargados de realizar acciones correctivas o de mantenimiento.

El sistema debe cumplir con las siguientes condiciones:

- Cada sensor genera un conjunto de tareas de inspección que deben ser enviadas a una cola centralizada de tareas compartida.
- Cada tarea debe representarse mediante un struct que contenga:
  - Un entero "idSensor", para identificar el sensor que la generó,
  - Un entero "idTarea", para distinguir cada tarea,
  - Una variable del tipo std::string, llamada "descripciónTarea", que describa la tarea (debe ser una descripción genérica que incluya el número de tarea).
- Cada tarea debe demorar 175 ms en ser creada.
- Un grupo de robots autónomos consume estas tareas de la cola, procesándolas en orden de llegada.
- Cuando la cola esté vacía, los robots deben esperar hasta que nuevas tareas estén disponibles.
- El procesamiento de cada tarea por parte de un robot debe tomar 250 ms.
- Una vez que todos los sensores hayan finalizado la generación de tareas, los robots deben terminar su trabajo luego de procesar todas las tareas pendientes.
- Se deben garantizar las condiciones de exclusión mutua y sincronización, evitando condiciones de carrera.

Se pide:

a. Implementar el sistema usando *std::threads* para representar a sensores y robots, y una cola de tareas compartida implementada como std::queue<Tarea>.
b. Utilizar los mecanismos de sincronización apropiados para coordinar sensores y robots: Sólo se permite usar std::condition_variable, std::unique_lock, *std::lock_guard* y *std::mutex*.
c. Crear un caso de prueba con 3 sensores y 3 robots, y mostrar por consola las acciones de generación y procesamiento de tareas para poder seguir su funcionamiento.

Dado que el ejercicio trata de evaluar al estudiante en su manejo de programación multihilo, está permitido que en este ejercicio utilice variables globales. Al mismo tiempo, puede incluir solo para este ejercicio su código en un único archivo .CPP.

Al igual que en el ejercicio 2, en la resolución de su ejercicio, no debe observarse que los mensajes por pantalla se superpongan en ningún momento. Tampoco deben existir tiempos donde los threads duermen que no sean los de 175 ms y 250 ms.