Asignación #1 – Video Game Inventory Engine (C++)

Curso: Introducción a Estructuras de Datos
Lenguaje: C++
Modalidad: Trabajo individual

Descripción general

En muchos videojuegos (RPG, survival, shooters), el inventario es un componente central del sistema. No se trata únicamente de almacenar objetos, sino de manejar restricciones como la capacidad, el peso total, los duplicados y operaciones básicas de consulta y modificación.

En esta asignación implementarán el **motor de inventario (backend)** de un videojuego. No se requiere interacción con el usuario ni una interfaz gráfica. El enfoque está en el **diseño de clases**, **uso de estructuras de datos**, **validaciones** y **buenas prácticas en C++**.

La evaluación se realizará mediante **casos de prueba automatizados** proporcionados por el instructor.

Objetivos de aprendizaje

- Diseñar clases bien encapsuladas en C++
- Separar correctamente archivos de definición (`.h`) e implementación (`.cpp`)
- Utilizar `std::vector` para almacenar colecciones de objetos
- Implementar validaciones que mantengan el estado consistente
- Preparar código para evaluación automatizada
- Documentar un proyecto de programación de forma clara en GitHub

---

Componentes del sistema

Clase `Item`

Representa un objeto que puede almacenarse en un inventario de videojuegos.

Cada ítem debe contener, como mínimo:

- nombre (`std::string`)
- peso (`int`, mayor o igual que 0)
- valor (`int`, mayor o igual que 0)
- tipo (`ItemType`)
- poder (`int`, mayor o igual que 0)

El tipo del ítem debe representarse mediante un `enum`, por ejemplo:

```
C/C++
enum ItemType { HEALING, WEAPON, ARMOR, MISC };
```

Clase Inventory

Representa el almacenamiento de ítems y aplica restricciones típicas de videojuegos.

El inventario tiene las siguientes restricciones:

- número máximo de ítems (`maxSlots`)
- peso total máximo permitido (`maxWeight`)

El inventario debe permitir:

- añadir ítems si hay espacio y peso disponible
- remover ítems por nombre
- verificar si un ítem existe
- consultar el peso total actual
- consultar la cantidad de ítems almacenados

No se requiere mantener un orden específico en el inventario.
Se permiten ítems con nombres duplicados; las operaciones deben afectar el **primer ítem encontrado**.

Visibilidad de atributos y métodos

Para evitar ambigüedades, se espera el siguiente diseño general.

Clase Item

Atributos (`private`):

- nombre

- peso
- valor
- tipo
- poder

Métodos (`public`):

- constructor
- getters (opcionales)

Los atributos no deben ser públicos.

Clase Inventory

Atributos (`private`):

- `std::vector<Item>` para almacenar ítems
- `maxSlots`
- `maxWeight`

Métodos (`public`):

- métodos especificados en la interfaz obligatoria
- métodos de consulta del estado del inventario

Métodos auxiliares:

- Pueden ser públicos o privados
- No se evalúan
- No deben producir entrada/salida

Interfaz obligatoria

Los siguientes elementos **forman parte de la interfaz pública** y **deben existir exactamente como se especifica** (nombre, parámetros y tipo de retorno).
El sistema de evaluación automática utilizará únicamente esta interfaz.

```C/C++
// Inventory
Inventory(int maxSlots, int maxWeight);

bool addItem(const Item& item);
bool removeItem(const std::string& name);
bool hasItem(const std::string& name) const;

int getCurrentWeight() const;
int getItemCount() const;
int getMaxSlots() const;
int getMaxWeight() const;

// Item
Item(std::string name, int weight, int value, ItemType type, int power);
```

- El constructor de `Inventory` inicializa un inventario vacío con los límites indicados. No es necesario implementar constructores adicionales.
- Puedes implementar métodos auxiliares adicionales si lo consideras necesario. Estos métodos no afectan la calificación.

Reglas y validaciones

- Todos los métodos deben validar sus parámetros.

- Ante una entrada inválida, el método debe:
  - retornar `false`
  - no modificar el estado del objeto

- Si el inventario está lleno o se excede el peso máximo permitido, la operación falla.
- Las comparaciones de nombres son **case-sensitive**.
- Todas las operaciones usan valores de retorno `bool` para indicar éxito o fallo.

Restricciones importantes

- No incluir `main()` en los archivos entregados.
- No usar `cin`, `cout` ni ningún tipo de entrada ni salida.
- No leer ni escribir archivos.
- Se permite el uso de `std::vector`.

- Usar correctamente los archivos `.h` y `.cpp`.

Puedes crear un `main.cpp` localmente para pruebas personales, pero **no debe subirse al repositorio**.

Evaluación automática

Tu código será compilado y evaluado usando un archivo `tests.cpp` del instructor, de forma equivalente a:

```
g++ -std=c++17 -Wall -Wextra Item.cpp Inventory.cpp tests.cpp
```

Cambiar los nombres de métodos, las firmas de método o el comportamiento esperado puede hacer que los casos de prueba fallen.

Archivos a entregar

El repositorio debe contener únicamente:

- `Item.h`
- `Item.cpp`
- `Inventory.h`
- `Inventory.cpp`
- `README.md`

README.md

El archivo `README.md` debe incluir:

- una descripción breve del proyecto
- una explicación general del diseño
- instrucciones de compilación
- nombre del estudiante

Criterios de evaluación

Componente	Porcentaje
Casos de prueba automáticos	70%
Diseño y validaciones	20%
Organización del código	5%
README y repositorio	5%

Nota final

El manejo adecuado de los inventarios es un problema real en el desarrollo de videojuegos. En proyectos profesionales, este componente interactúa con múltiples sistemas (UI, economía, combate, guardado de partidas). Por esta razón, la claridad, la encapsulación y la consistencia del estado son esenciales.