Repaso: Examen Estantería 2021–22

MP Ordinaria – 24 de Junio de 2022

Solución del profesor simplificada para repaso

Contexto: estructura de clases

```
class Libro  { string id, autor, titulo; Fecha fecha_compra; int grosor; };
class Estante { int largo; Libro* libros; int nLibros; };      // <- Big Four propio
class Estanteria { int largo; Estante* estantes; int nEstantes; }; // <- Big Four dado
```

Diferencia clave con otros examenes

El capacidad de `Estante` no dobla — está limitada físicamente por `largo`. Por eso `+=` crea un array de `nLibros+1` exacto (sin doblar). Además, los libros deben mantenerse **ordenados por fecha de compra**.

Ejercicio 1 – Big Four de Estante [1 pt]

Enunciado Ej1

Implemente los siguientes métodos de la clase `Estante`:

1. **Constructores** (sin parámetros y con un parámetro entero). Ambos crean un estante sin libros. El constructor con parámetros recibe un valor entero para establecer el largo.
2. **Destructor**. ¿Hace falta? Justifique y en caso afirmativo, implémentelo.
3. **Constructor de copia y operador de asignación**. ¿Hace falta implementarlos? Justifique y en caso afirmativo, implémentelos.

```
// ------- HELPERS PRIVADOS -------
void Estante::inicializar(int _largo) {
    largo   = (_largo >= 0) ? _largo : 0;
    nLibros = 0;
    libros  = nullptr;
}
void Estante::liberarMemoria() {
    delete[] libros;
    libros = nullptr;      // siempre anular tras liberar
}
void Estante::reservarMemoria(int n) {
    if (n > 0) { nLibros = n; libros = new Libro[n]; }
    else          inicializar();
}
void Estante::copiar(const Estante& otro) {
    largo   = otro.largo;
    nLibros = otro.nLibros;
    for (int i = 0; i < nLibros; i++)
        libros[i] = otro.libros[i];
}

// ------- CONSTRUCTORES -------
Estante::Estante(int _largo) {       // cubre Estante() con valor por defecto 0
    inicializar(_largo);
}
Estante::~Estante() {
    liberarMemoria();
}
Estante::Estante(const Estante& otro) {
    reservarMemoria(otro.nLibros);
    copiar(otro);
}
Estante& Estante::operator=(const Estante& otro) {
```

```
    if (this != &otro) {
        liberarMemoria();
        reservarMemoria(otro.nLibros);
        copiar(otro);
    }
    return *this;
}
```

Por que hace falta el Big Four

`Estante` tiene un puntero `Libro* libros` con memoria dinámica ⇒ destructor para no fugar memoria, copia profunda para no compartir punteros entre objetos.

Patron: reservarMemoria + copiar

El constructor de copia siempre: (1) reserva espacio, (2) copia. El `operator=` siempre: (1) comprueba autoasignación, (2) libera, (3) reserva, (4) copia.

Ejercicio 2 – Operadores += y -= para Estante [2.5 pt]

Enunciado Ej2

Sobrecargue los operadores combinados `+=` y `-=` para la clase `Estante`.

- `+=`: Incorpora un nuevo `Libro` al estante **si cabe** (si su grosor es menor que el espacio disponible). Los libros se mantienen **ordenados por fecha de compra**.
- `-=`: Elimina un libro dado su identificador `id`, desplazando el resto hacia la izquierda y ajustando el array.

```
// --- HELPER: espacio ocupado por los libros actuales ---
int Estante::espacioOcupado() const {
    int total = 0;
    for (int i = 0; i < nLibros; i++)
        total += libros[i].getGrosor();
    return total;
}

// --- HELPER: buscar libro por id, devuelve posicion o -1 ---
int Estante::buscar(const string& id) const {
    for (int i = 0; i < nLibros; i++)
        if (libros[i].getId() == id) return i;
    return -1;
}

// --- OPERADOR += ---
Estante& Estante::operator+=(const Libro& libro) {
    if ((largo - espacioOcupado()) <= libro.getGrosor())
        return *this;   // no cabe, no hacemos nada

    Libro* nuevo = new Libro[nLibros + 1];

    // Copiar los que van ANTES del libro nuevo (fecha menor)
    int i = 0;
    while (i < nLibros && libros[i].getFechaCompra() < libro.getFechaCompra()) {
        nuevo[i] = libros[i];
        i++;
    }
    nuevo[i] = libro;   // insertar en su posicion
    // Copiar el resto (van detras)
    for (; i < nLibros; i++)
        nuevo[i + 1] = libros[i];

    nLibros++;
    delete[] libros;
```

```
    libros = nuevo;
    return *this;
}

// --- OPERADOR -= ---
Estante& Estante::operator-=(const Libro& libro) {
    int pos = buscar(libro.getId());
    if (pos < 0) return *this;   // no existe

    Libro* nuevo = new Libro[nLibros - 1];
    for (int i = 0; i < pos; i++)      nuevo[i]   = libros[i];
    for (int i = pos; i < nLibros-1; i++) nuevo[i] = libros[i + 1];

    nLibros--;
    delete[] libros;
    libros = nuevo;
    return *this;
}
```

La condicion de += es estricta

El enunciado dice "si el grosor es **menor** que el espacio disponible". Ojo: `menor`, no menor o igual. Por tanto: `(largo - espacioOcupado()) <= libro.getGrosor()` ⇒ no cabe.

+= no dobla capacidad

A diferencia de los otros exámenes (Tienda, Catálogo), aquí se crea un array de exactamente `nLibros+1` porque la capacidad está limitada por `largo`, no por una capacidad abstracta. Igual para `-=`: array de `nLibros-1`.

Ejercicio 3 – E/S para Estanteria [2.5 pt]

Enunciado Ej3

3.1 Sobrecargue `«` y `»` para la clase `Estanteria` con el formato:

```
ESTANTES: 3
LARGO: 186
2
PDU00; Philip K. Dick; Ubik; 02/10/2000; 4;
GO120; George Orwell; 1984; 10/04/2020; 5;
...
```

3.2 Constructor desde fichero: primera línea debe ser `"DATOSESTANTERIA"`, resto tiene el formato anterior. Si la cadena no coincide, crear estantería vacía.

```
// --- operator<< para Estante (auxiliar) ---
ostream& operator<<(ostream& flujo, const Estante& e) {
    flujo << e.nLibros << "\n";
    for (int i = 0; i < e.nLibros; i++)
        flujo << e.libros[i] << "\n";   // << de Libro ya disponible
    return flujo;
}

// --- operator<< para Estanteria ---
ostream& operator<<(ostream& flujo, const Estanteria& est) {
    flujo << "ESTANTES: " << est.nEstantes << "\n";
    flujo << "LARGO: "    << est.largo     << "\n";
    for (int i = 0; i < est.nEstantes; i++)
        flujo << est.estantes[i];       // usa << de Estante de arriba
    return flujo;
}

// --- operator>> para Estanteria ---
```

```cpp
istream& operator>>(istream& flujo, Estanteria& est) {
    string etiqueta;
    int nEstantes, largo;

    flujo >> etiqueta >> nEstantes;    // "ESTANTES:" N
    flujo >> etiqueta >> largo;        // "LARGO:" L
    est.setLargo(largo);
    est.liberarMemoria();
    est.reservarMemoria(nEstantes);    // reserva y asigna largo a cada estante

    for (int i = 0; i < nEstantes; i++)
        flujo >> est.estantes[i];      // usa >> de Estante
    return flujo;
}

// --- Constructor desde fichero ---
Estanteria::Estanteria(const string& nombreFichero) {
    inicializar();                     // empieza vacio
    ifstream fichero(nombreFichero);
    if (fichero) {
        string cadena;
        getline(fichero, cadena);      // lee primera linea
        if (cadena == "DATOSESTANTERIA")
            fichero >> *this;          // reutiliza operator>>
        fichero.close();
    }
    // si no abre o cadena incorrecta -> queda vacio (inicializar ya lo hizo)
}
```

reservarMemoria de Estanteria asigna largo

Cuando `reservarMemoria(n)` crea los `n` estantes, les asigna `setLargo(largo)` a cada uno para que estén configurados correctamente antes de leer sus libros.

.

Ejercicio 4 – reorganizar() [2 pt]

Enunciado Ej4

Implemente el método `reorganizar` de la clase `Estanteria` que redistribuye los libros para optimizar el espacio (posiblemente reduciendo el número de estantes).
Sugerencia: "Volcar" todos los libros a un estante auxiliar de largo suficiente, luego procesar secuencialmente creando nuevos estantes.

```cpp
// --- Auxiliar: espacio total ocupado en toda la estanteria ---
int Estanteria::espacioOcupado() const {
    int total = 0;
    for (int i = 0; i < nEstantes; i++)
        total += estantes[i].espacioOcupado();  // usa el de Estante
    return total;
}

void Estanteria::reorganizar() {
    // PASO 1: volcar todos los libros en un unico estante auxiliar gigante
    Estante aux(espacioOcupado());   // largo = total ocupado -> todos caben
    for (int i = 0; i < nEstantes; i++)
        for (int j = 0; j < estantes[i].getNLibros(); j++)
            aux += estantes[i][j];   // += de Estante (inserta en orden por fecha)

    // PASO 2: limpiar la estanteria actual
    liberarMemoria();

    // PASO 3: redistribuir desde el aux creando nuevos estantes
```

```
    Estante estActual(largo);
    for (int i = 0; i < aux.getNLibros(); i++) {
        if ((largo - estActual.espacioOcupado()) <= aux[i].getGrosor()) {
            // no cabe en el estante actual -> lo cierra y abre uno nuevo
            (*this) += estActual;        // += Estante (ya disponible)
            estActual = Estante(largo); // nuevo estante vacio
        }
        estActual += aux[i];             // insertar libro en el estante actual
    }
    // anadir el ultimo estante si tiene libros
    if (estActual.getNLibros() > 0)
        (*this) += estActual;
}
```

Algoritmo en dos pasos

Paso 1: Volcar (todos los libros → estante aux). **Paso 2**: Redistribuir (recorrer aux, llenar estante hasta que no quepa, añadir a la estantería, abrir uno nuevo).

liberarMemoria antes de redistribuir

Es imprescindible llamar a `liberarMemoria()` entre los pasos 1 y 2. Si no, la estantería sigue con los estantes viejos y al hacer `+=` se añaden estantes al final en lugar de empezar desde cero.

Ejercicio 5 – main: estantería de repetidos [2 pt]

Enunciado Ej5

Programa que recibe dos ficheros de estanterías y un entero (largo destino). Crea `e3` con los libros que aparecen en ambas estanterías y la guarda en `repetidos.txt`.
`argv[1]` = fichero estantería 1, `argv[2]` = fichero estantería 2, `argv[3]` = largo de `e3`.

```
// --- Auxiliar en Estanteria: buscar libro por id en todos los estantes ---
bool Estanteria::buscar(const string& id) const {
    for (int i = 0; i < nEstantes; i++)
        if (estantes[i].buscar(id) != -1)  // buscar() de Estante
            return true;
    return false;
}

// --- Auxiliar en Estanteria: salvar en fichero ---
void Estanteria::salvar(const string& nombreFichero) const {
    ofstream f(nombreFichero);
    if (f) { f << "DATOSESTANTERIA\n" << *this; }
}

int main(int argc, char* argv[]) {
    Estanteria e1(argv[1]);
    Estanteria e2(argv[2]);
    int largo = atoi(argv[3]);
    Estanteria e3(largo);

    Estante estAux(largo);
    for (int i = 0; i < e1.getNEstantes(); i++) {
        for (int j = 0; j < e1[i].getNLibros(); j++) {
            if (e2.buscar(e1[i][j].getId())) {      // existe en e2?
                if ((largo - estAux.espacioOcupado()) <= e1[i][j].getGrosor()) {
                    e3 += estAux;                   // estante lleno -> anadir
                    estAux = Estante(largo);
                }
                estAux += e1[i][j];
            }
        }
    }
```

```
    if (estAux.getNLibros() > 0)
        e3 += estAux;

    e3.salvar("repetidos.txt");
    return 0;
}
```

Patron main con estante auxiliar

El mismo patrón del Ej4: acumular libros en `estAux`, cuando no cabe → añadir el estante a la estantería y abrir uno nuevo. Al final, añadir el último si no está vacío.

Tabla resumen de patrones

Concepto	Este examen	Otros exámenes
Crecer array	`new[nLibros+1]` exacto	`new[cap*2]` (doblar)
Reducir array	`new[nLibros-1]` exacto	Shift + `usados--`
Inserción en `+=`	Ordenada por fecha	Al principio/final
Capacidad máx.	Física: `largo`	Lógica: `capacidad`
Cadena mágica	`"DATOSESTANTERIA"`	`"TIENDA_ORLAS"` / `RUTAS"`
Ej4 algoritmo	Volcar + redistribuir	Ordenar / filtrar

Los 3 errores mas frecuentes en este examen

1. `+=` Estante: olvidar el `nLibros++` después de crear el nuevo array.
2. `-=` Estante: el segundo bucle empieza en `pos`, no en `pos+1`: `nuevo[i] = libros[i+1]` (la indexación ya compensa).
3. `reorganizar`: olvidar `liberarMemoria()` entre volcar y redistribuir.