1. Cree una clase **Counted** que contenga un entero **id** y un entero estático **count**. El constructor debería empezar: **Counted( ) : id(count++) {.** Debería también imprimir su **id** y que está siendo creado. El destructor debería imprimir que se está destruyendo. Pruebe la clase.
2. Compruebe que **new** y **delete** siempre llaman a los constructores y destructores sobre instancias de **Counted**. Pruebe también creando arreglos de estos objetos sobre el heap.
3. Cree un **vector<Counted*>** y llénelo con punteros a instancias de **Counted**. Recorra el vector imprimiendo los objetos y recórralo nuevamente llamando a **delete** sobre cada uno.
4. Cree dinámicamente un arreglo de instancias de **Counted**. Llame a **delete** para el puntero resultante sin usar corchetes. Explique el resultado.
5. Cree un objeto de la clase **Counted** usando **new**, castee el puntero resultante a un **void*** y llame a **delete** sobre él. Explique el resultado.
6. Implemente una lógica para calcular la cantidad de memoria que puede utilizar del heap (utilice memory exhaustion).
7. Cree una clase con los operadores **new** y **delete** sobrecargados, ambos simples y versión arreglo. Demuestre que ambas versiones funcionan.
8. Suponga que está diseñando una clase que será instanciada múltiples veces, pero que por diseño, nunca habrá más de N instancias simultáneas. Sobrecargue los operadores **new** y **delete** para esta clase para que utilicen un mecanismo eficiente de alocación (sin fragmentación y con costo constante).
9. Crear e implementar clases **Rueda**, **Motor**, **Puerta**, **Ventanilla**, **Auto** y **Vehiculo** con métodos y atributos apropiados. Discutir posibilidades de implementación vía composición y/o herencia.
10. Crear una jerarquía de 4 clases, 3 derivadas y una embebida. Dotarlas de destructores y constructores que se anuncien en **cout**. Implementar un **main** utilizando objetos de estas clases y verificar orden de llamado de constructores y destructores.
11. Crear dos clases llamadas **Traveler** y **Pager** con constructores que reciben como argumento un **string** que se copia a un atributo privado. Generar constructores y operadores de copia y move para cada una. Crear una clase **BusinessTraveler** que herede de **Traveler** y contenga un objeto **Pager** como atributo. Generarle un constructor default, un constructor que reciba 2 **string**s, constructores y operadores de copy y move. Hacer un programa para probar lo implementado.
12. Crear una clase **SpaceShip** con un método **fly()**. Crear una clase **Shuttle** derivada de **SpaceShip** con un método **land()**. Crear un objeto **Shuttle**, hacer un upcast a puntero o referencia a **SpaceShip** y llamar al método **land()** sobre el upcast. Explicar.
13. Crear una clase **Cosa** con constructor default, constructores y operadores de copy y move, y destructor cuyas implementaciones se anuncien en **cout**. Hacer un programa donde haya un **vector<Cosa>**, agregar varias **Cosa**s y explicar el output. Repetir con un **vector<Cosa *>**.