1. Escribir un programa que pruebe que el compilador sintetiza automáticamente un constructor de copia si no se define uno. Pruebe que el constructor sintetizado realiza un bitcopy de los tipos primitivos y llama a constructores de copia para tipos definidos por el usuario. (*)
2. Escriba una clase con un constructor de copia que se anuncie a sí mismo en **`cout`**. Ahora cree una función que pasa un objeto de su nueva clase por valor y otra que crea un objeto local de la nueva clase y lo retorna por valor. Pruebe que el constructor de copia es llamado cuando se pasan argumentos o se retornan objetos por valor.
3. Cree una clase que contiene un **`double*`**. El constructor inicializa el puntero a doble llamando a **`new double`** y asigna como contenido del puntero al argumento del constructor. El destructor imprime el valor que contiene el puntero, asigna al valor -1, llama a **`delete`** para liberar el espacio y pone el puntero en cero. Ahora cree una función que tome un objeto de su clase por valor y llame a esta función en **`main( ).`** ¿Qué pasa? ¿Cómo arreglaría el problema?

los dos punteros apuntan ala misma direccion de memoria

4. Cree una clase con un constructor que sea similar a un constructor de copia pero que reciba un argumento adicional con valor por defecto. Pruebe que todavía es utilizado como constructor de copia.

pq pasa?

si no le pongo por defecto, queda esperando y usa el de copia definido por el compilador, no el que defini yo

5. Cree una clase con un constructor de copia que se anuncie a sí mismo. Haga una segunda clase conteniendo un objeto miembro de la clase anterior pero no cree un constructor de copia. Muestre que el constructor sintetizado en la segunda clase llama automáticamente el constructor de copia de la primera clase.
6. Cree una clase conteniendo un **`double`** y una función **`print( )`** que imprime el **`double`**. En **`main( ),`** cree punteros a miembros para el dato y la función miembro. Cree un objeto de su clase y un puntero a ese objeto y manipule esos elementos a través de punteros a miembro utilizando tanto el objeto como el puntero.
7. Cree una clase haga anunciación de construcción y destrucción. Haga que la clase contenga como miembros instancias de otras clases que también hagan anunciación de construcción y destrucción. Verifique el orden en que se crean y destruyen los objetos.
8. Creen una clase que contenga move constructor y se anuncie. Haga operaciones de tal manera que aparezcan objetos temporarios de esa clase. Verifique la activación de los move constructors.
9. Cree una clase **`Manager`** con un constructor privado y cree una función estática pública: **`Manager *getInstance()`** que cree y/o retorne una única instancia de **`Manager`**. Pruebe que no es posible construir instancias directamente.

La asignacion de 0 a la variable static debo ponerla en el cpp no en el .h pq sale como multiple definition

10. Cree una clase **`A`** que contenga como miembros instancias de las clases **`B`** y **`C.`** Ni **`B`** ni **`C`** tienen constructores por defecto. Compruebe que la única forma de construir los miembros de **`B`** y **`C`** de **`A`** es a través de la lista de inicialización. ¿Cómo justifica este comportamiento?

9) como el contructor es privado y get instance puede ser llamado solo una vez, van a tener todos los objetos la misma direccion de memoria.

10) si le hago delete a los constructores de las clases B y C, cuando creo un objeto de la clase A me sale un error de compilacion: se usa el constructor eliminado d elas clases B y C.