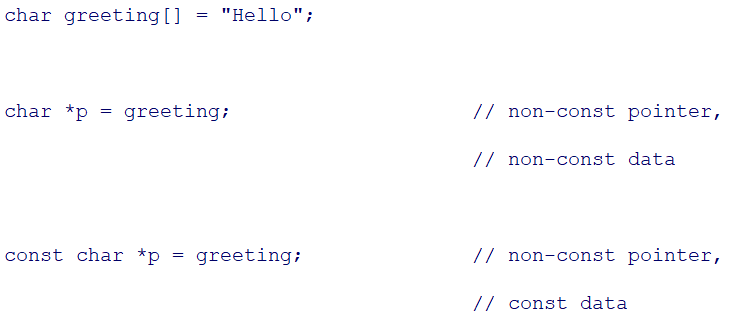
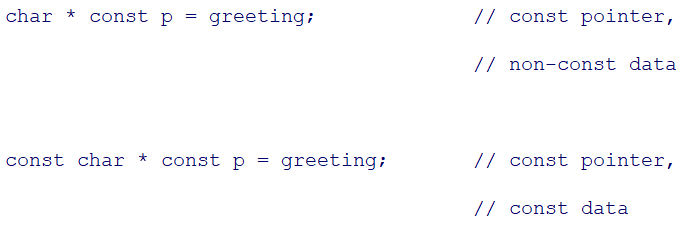
Item 3 Usar const siempre que sea posible

Lo maravilloso acerca de const es que le permite a usted especificar una restricción semántica a un objeto particular que no debe ser modificado y los compiladores forzarán esa restricción. El uso de const le permite a usted notificar a compiladores y programadores que un valor debe permanecer invariante. Siempre que eso sea cierto, usted debe asegurarse de decir que así es, porque de esa forma usted enlista la ayuda de sus compiladores para asegurar que la restricción no es violada.

La palabra reservada const es muy versátil. Fuera de las clases, usted puede usarla para constantes en alcance global o de espacio de nombres (Item 2 de [Meyers]), así como objetos declarados static en alcances de archivo, de función, o de bloque. Dentro de las clases, usted puede usarla con ambos, atributos estáticos y no estáticos. Para apuntadores, usted puede especificar si el apuntador mismo es constante, el dato al que apunta es constante, ambos son constantes, o ninguno es constante:





Esta sintaxis no es tan caprichosa como podría parecer. Si la palabra const aparece a la izquierda del asterisco, lo que es apuntado es constante; si la palabra const aparce a la derecha del asterisco, el apuntador mismo es constante; si const aparece a ambos lados, ambas cosas son constantes.

Cuando lo que es apuntado es constante, algunos programadores ponen const antes del tipo de dato. Otros lo ponen después del tipo de dato, pero antes del asterisco. No hay diferencia en el significado, así que las siguientes funciones toman el mismo tipo de parámetro:





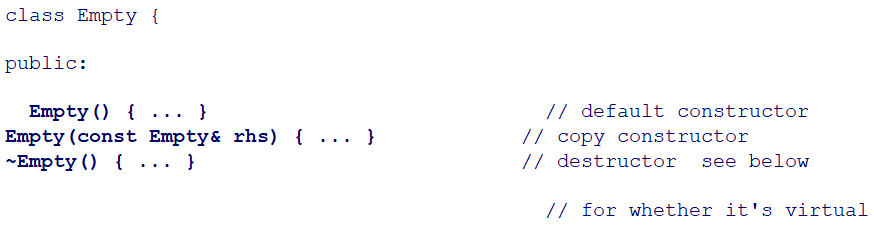
Dado que ambas formas existen en código real, usted debe acostumbrarse a ambas.

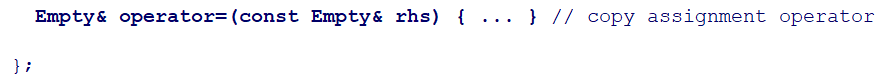
Item 5: Saber qué funciones C++ silenciosamente escribe y llama ([Meyers, 116,0/714])

¿Cuándo una clase vacía no es una clase vacía? Cuando C++ se encarga de ella. Si usted mismo no los declara, los compiladores declararán sus propias versiones de un constructor de copia, un operador de asignación de copia, y un destructor. Además, si usted no declara ningún constructor, los compiladores también declararán un constructor por defecto por usted. Todas estas funciones serán públicas e inline. Como resultado, si usted escribe

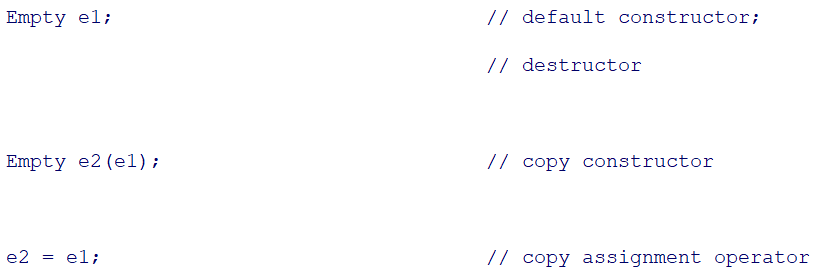


esto es esencialmente lo mismo que si usted hubiese escrito esto:





Esas funciones son generadas solamente si se necesitan, pero no se requiere hacer mucho para que sean necesarias. El siguiente código hará que cada una de las funciones sean generadas:



Item 3 Usted debe asegurarse de que los objetos son inicializados antes de que sean usados

Por ejemplo, si usted dice esto,



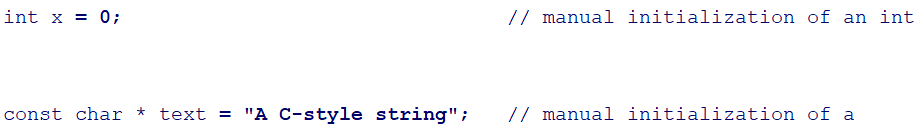
En algunos contextos, se garantiza que x será inicializada (a cero), pero en otros, no lo será. Si usted dice esto,

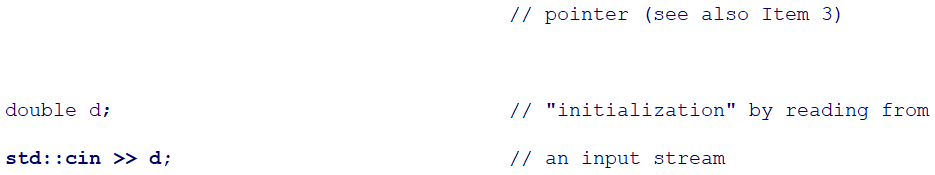




Los atributos de p algunas veces serán inicializados a cero, pero algunas veces no. Si usted viene de un lenguaje donde los objetos no inicializados no pueden existir ponga atención porque esto es importante.

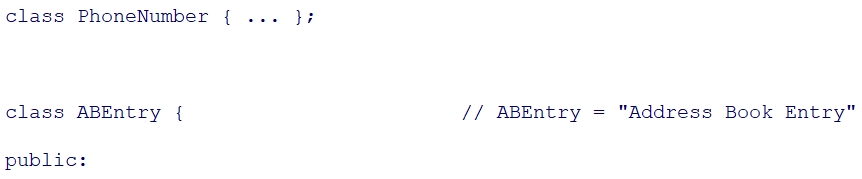
Para objetos que no son atributos de una clase, usted deberá inicializarlos manualmente. Por ejemplo:

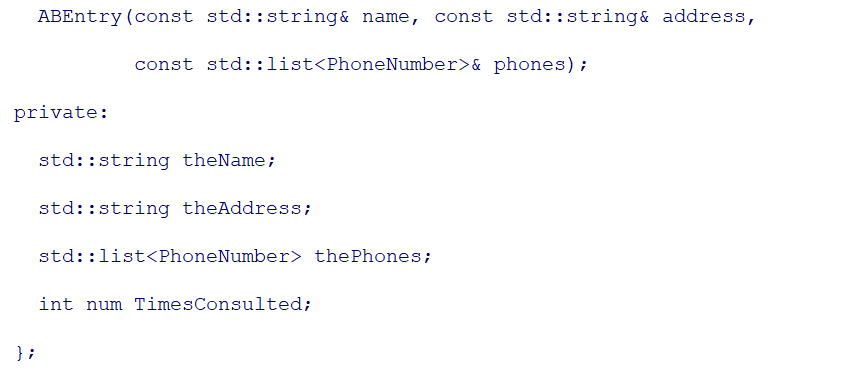


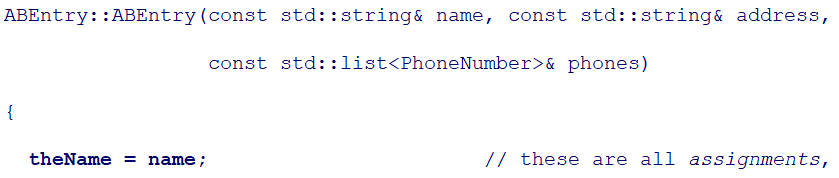


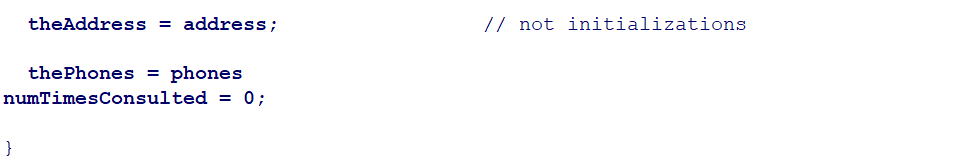
Para casi todo lo demás, la responsabilidad de inicialización recae en los constructores. La regla ahí es simple: asegúrese de que todos los constructores inicializan todo en el objeto.

La regla es fácil de seguir, pero es importante no confundir asignación con inicialización. Considere un constructor para una clase que representa entradas en un libro de direcciones:



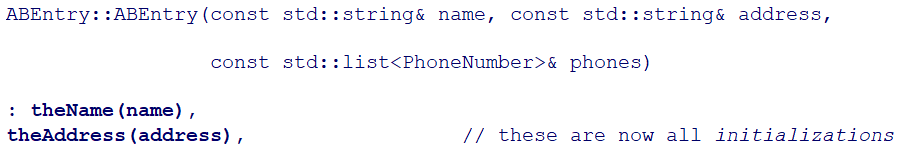






Esto producirá objetos ABEntry con los valores que usted espera, pero este no es aun el mejor enfoque. Las reglas de C++ estipulan que los atributos de un objeto son inicializados antes de que se introduzca el cuerpo de un constructor. Dentro del constructor ABEntry, theName, theAddress, y thePhones no están siendo inicializados, están siendo asignados. La inicialización tomó lugar antes cuando sus constructores por defecto fueron automáticamente llamados, previo a entrar al cuerpo del constructor ABEntry. Esto no es cierto para numTimesConsulted, porque su tipo de dato es un tipo de dato interno. Por eso, no hay garantía de que fue inicializado previo a su asignación.

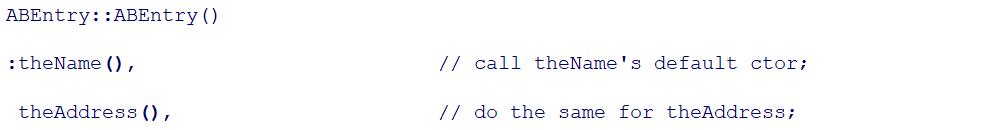
Una mejor forma de escribir el constructor ABEntry es usar la lista de inicialización de atributos en lugar de asignaciones:

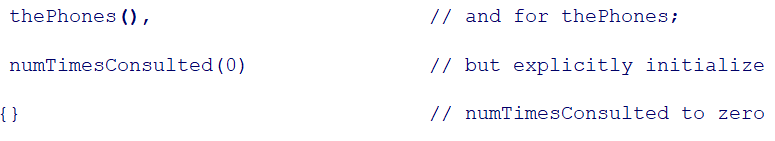




Este constructor produce el mismo resultado que el de arriba, pero frecuentemente es más eficiente. La primera versión basada en asignación llamó a los constructores por defecto para inicializar theName, theAddress, y thePhones, entonces rápidamente asignó nuevos valores sobre los construidos por defecto. Todo el trabajo realizado en esas construcciones por defecto fue por lo tanto desperdiciado. El enfoque de lista de inicialización de atributos evita ese problema, porque los argumentos en la lista de inicialización son usados como argumentos de constructor para los varios atributos. En este caso, theName, theAddress, thePhones son construidos con constructor de copia usando name, address, y phones respectivamente. Para la mayoría de los tipos, una sola llamada a un constructor de copia es más eficiente algunas veces mucho más eficiente que una llamada al constructor por defecto seguida de una llamada al operador de asignación de copia.

Para objetos de tipo interno como numTimesConsulted, no hay diferencia en costo entre inicialización y asignación, pero por consistencia, frecuentemente es mejor inicializar todo a través de inicialización de atributos. Similarmente, usted puede utilizar la lista de inicialización de atributos aun cuando usted quiera inicializar un atributo con el constructor por defecto; solo no se especifica algo como argumento de inicialización. Por ejemplo, si ABEntry tuviese un constructor que no recibe parámetros, éste podría ser implementado como sigue:





REF.

[Meyers] Effective C++, 3rd Edition, 55 Specific Ways to Improve Your Programs and Designs.