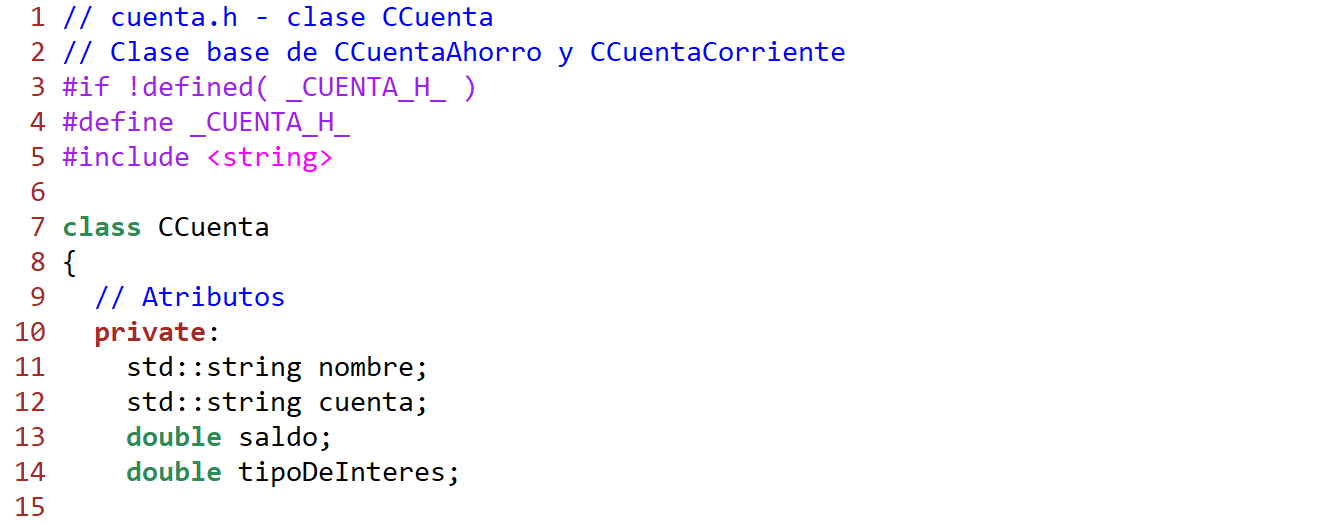
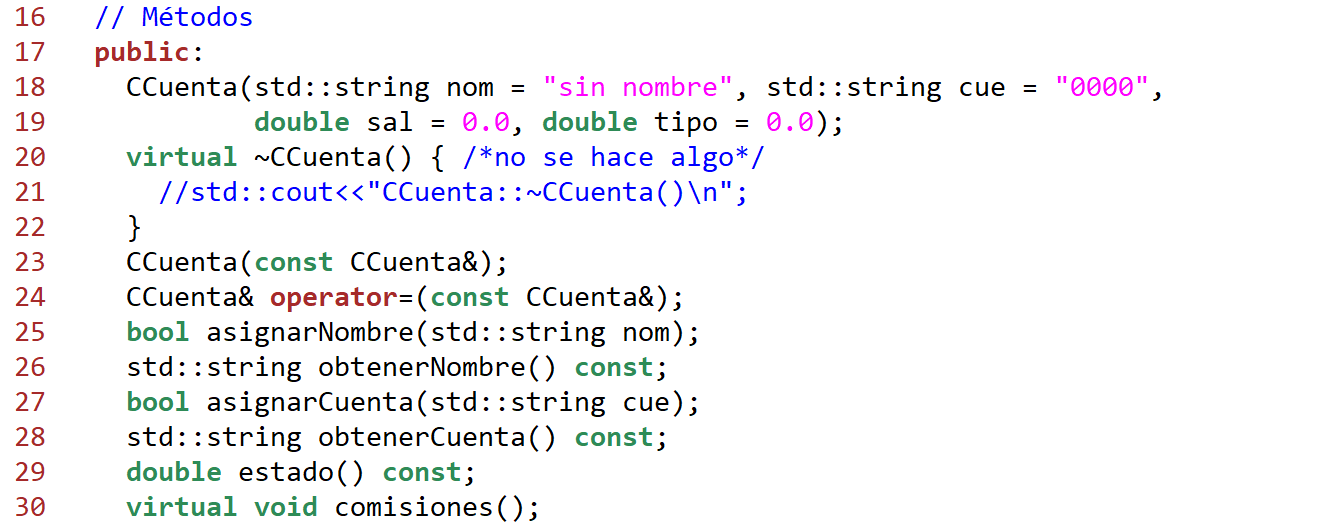
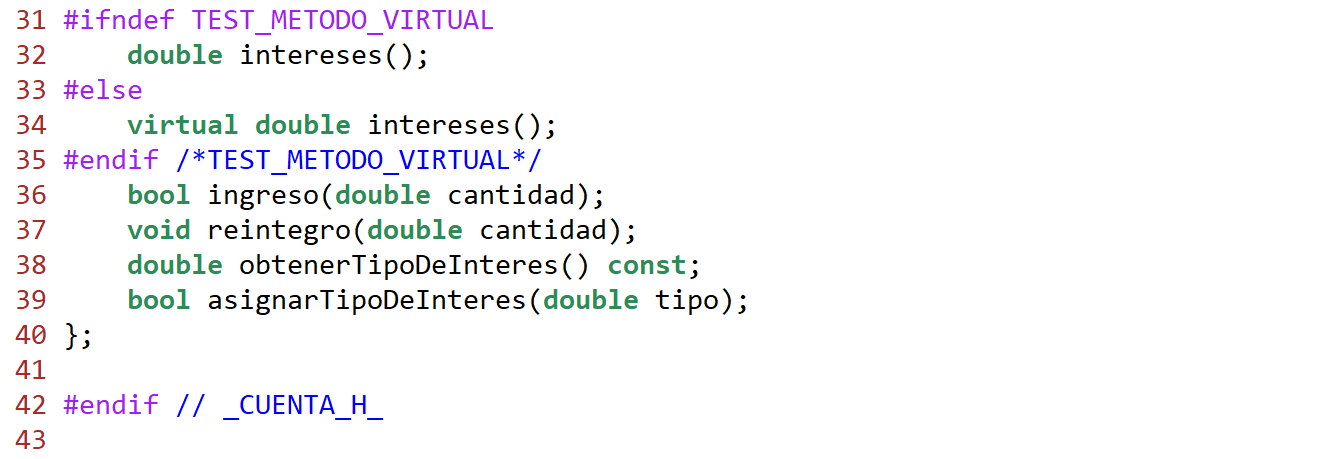
Métodos virtuales

Consideremos la clase CCuenta

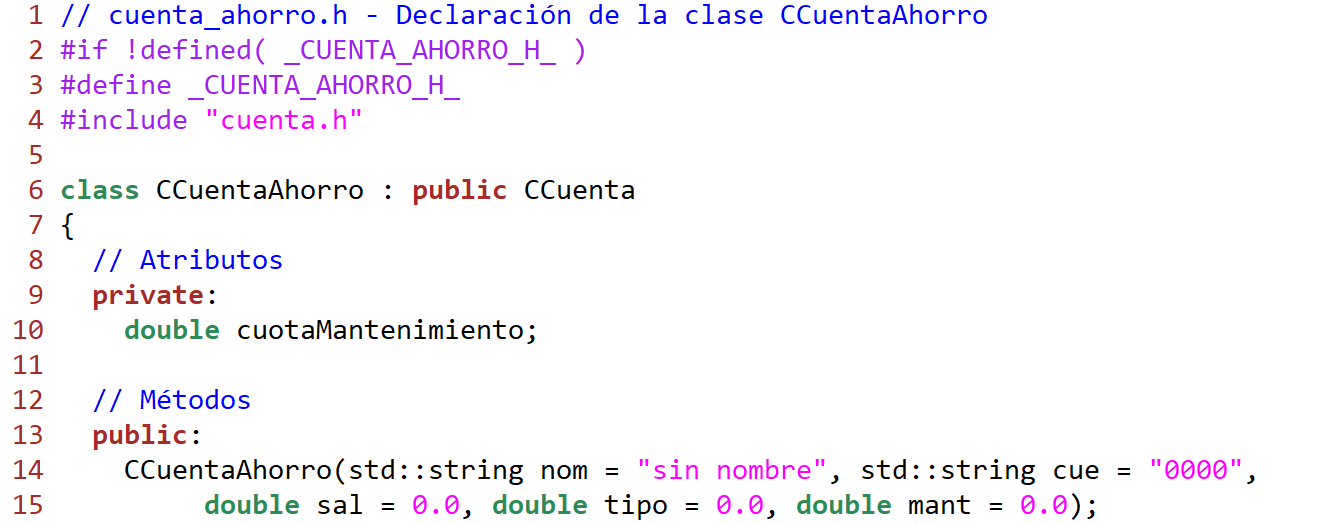


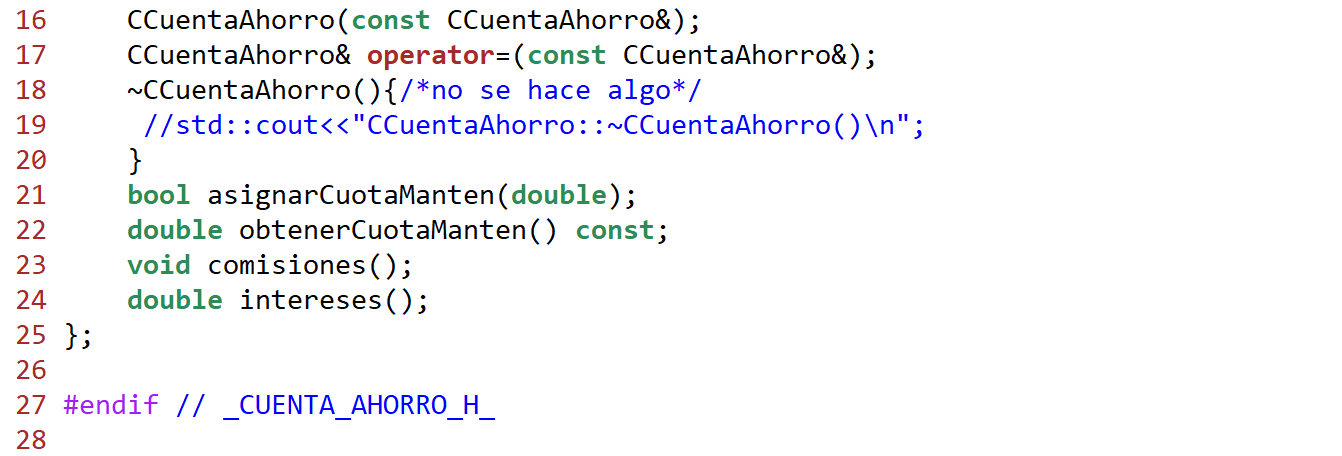




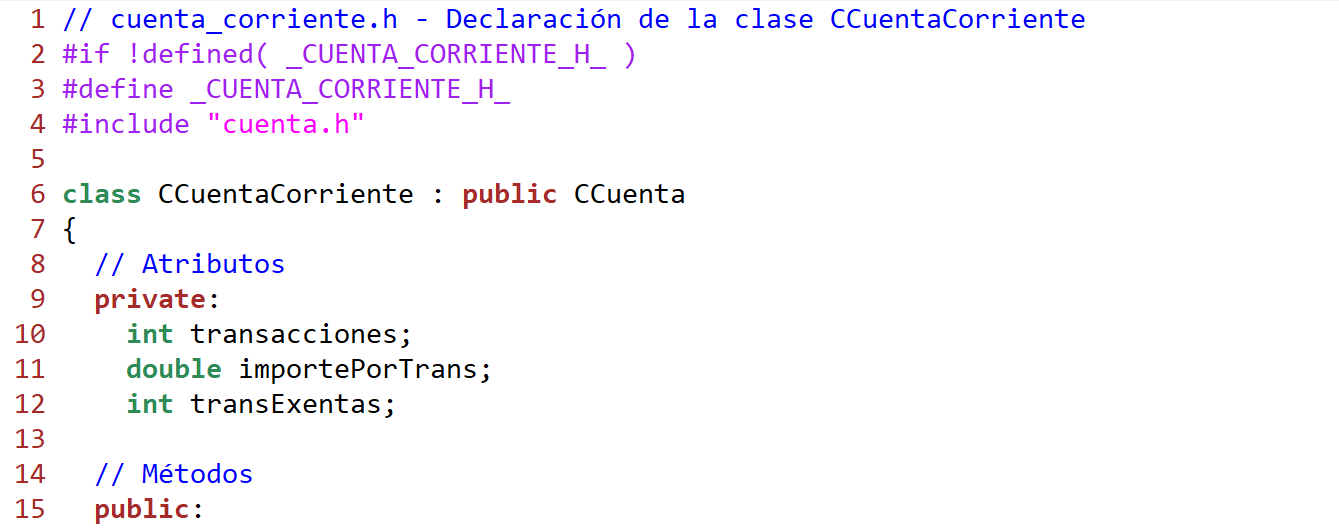
Archivo cuenta.h

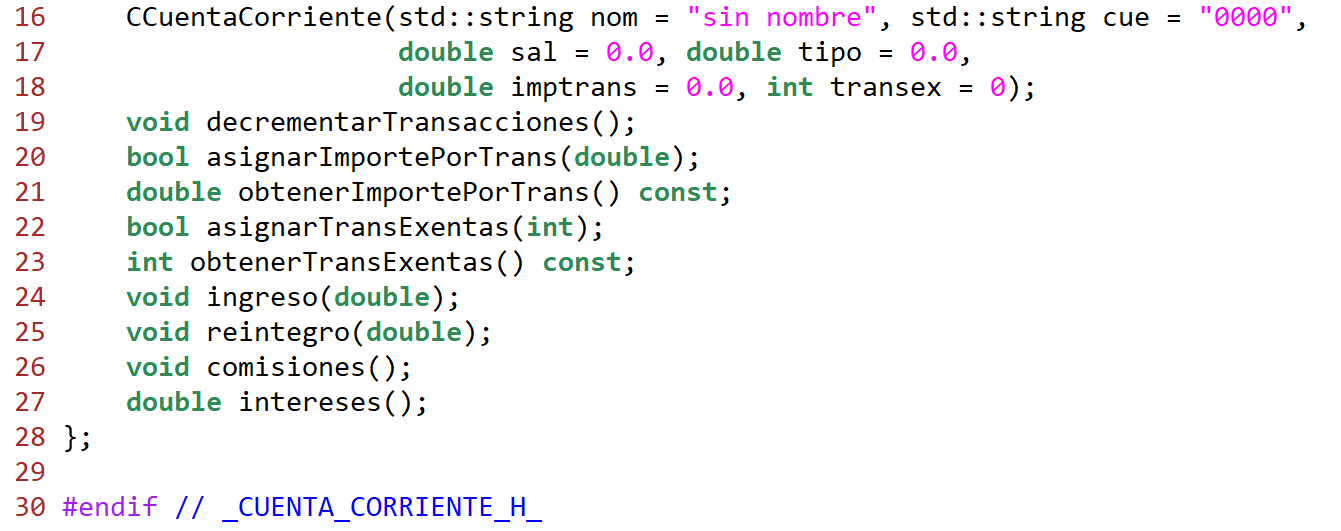
El método intereses() de la clase CCuenta se ha redefinido en todas sus clases derivadas, directas o indirectas (el método comisiones también, excepto en la clase CuentaCorrienteConIn). A continuación se incluyen las declaraciones de las clases derivadas y los códigos del método intereses() en las clases base y derivadas.



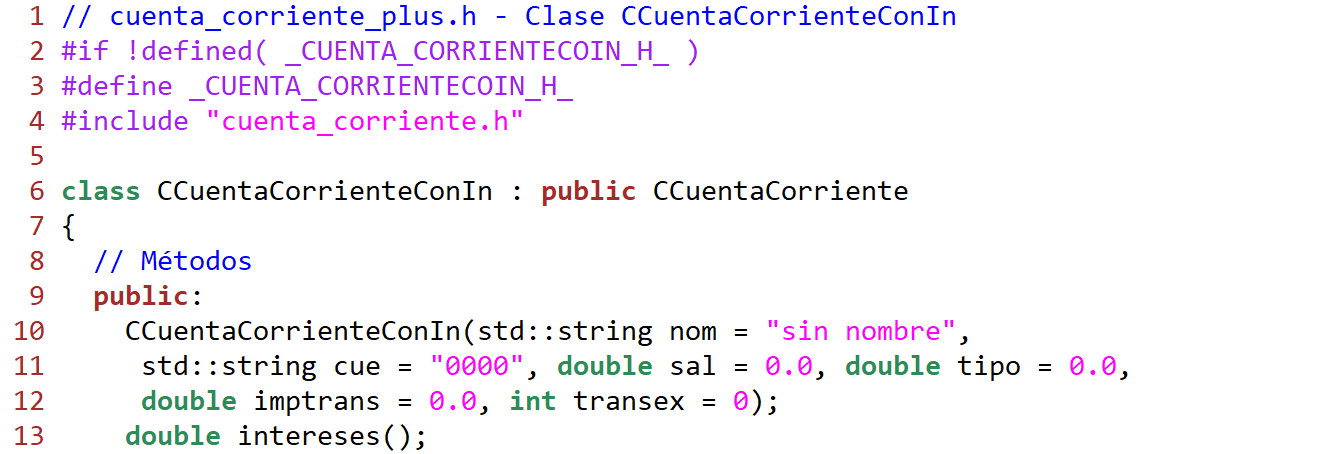


Archivo cuenta\_ahorro.h





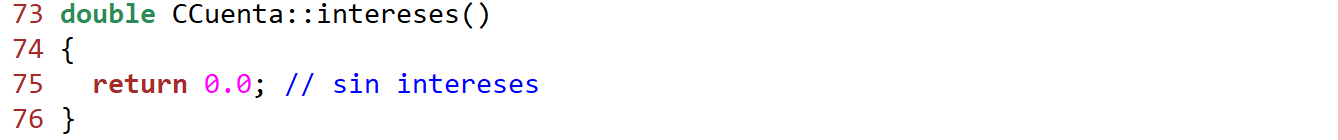
Archivo cuenta\_corriente.h





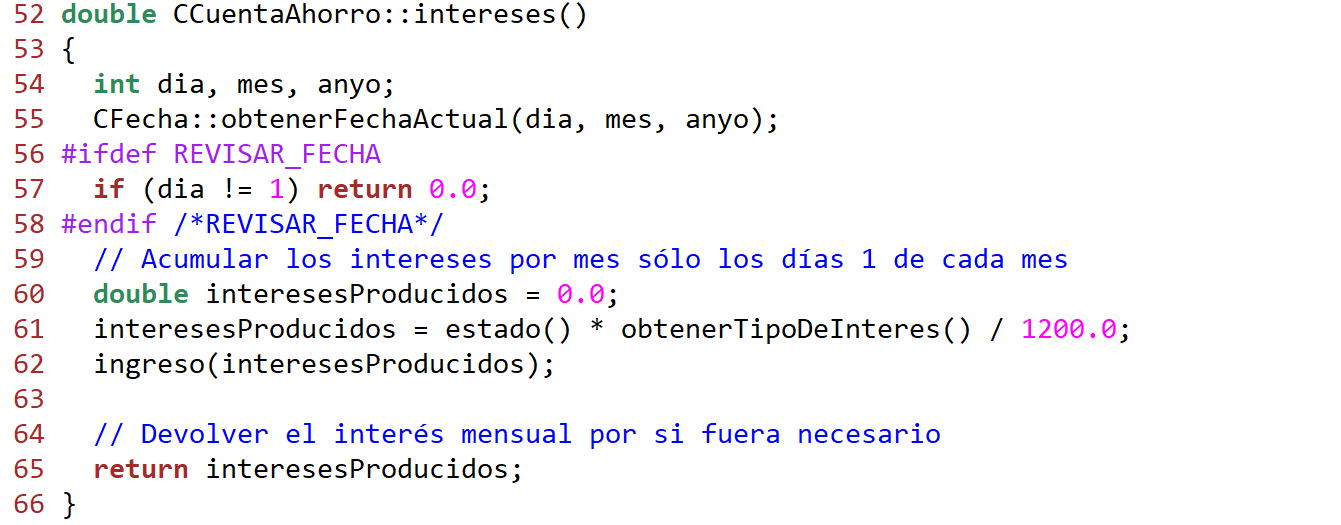
Archivo cuenta\_corriente\_plus.h

El método intereses() de la clase CCuenta



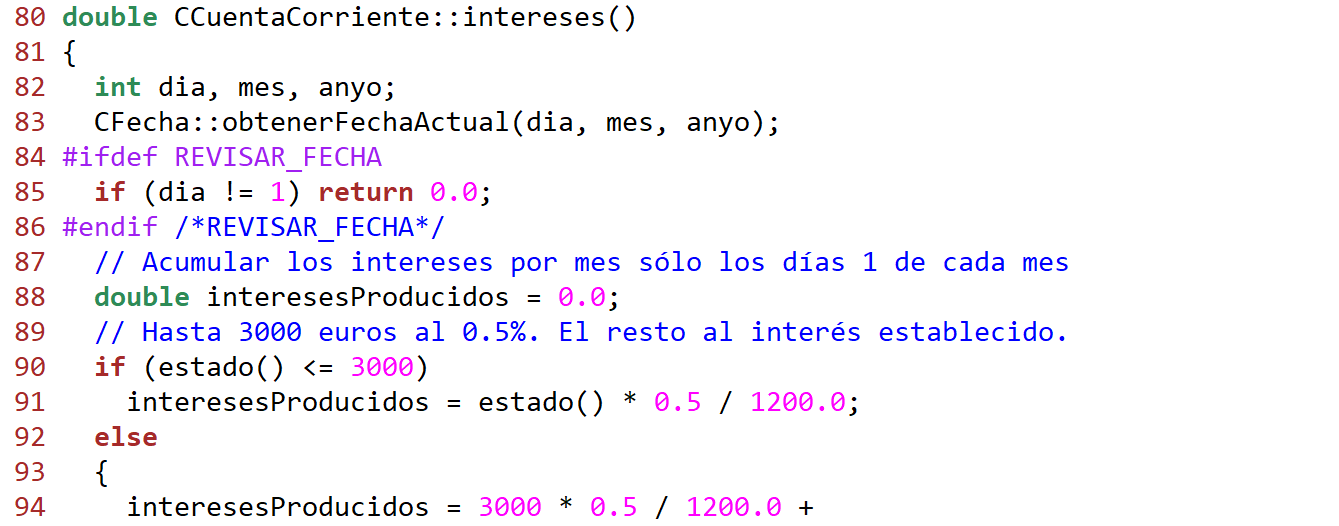
Fragmento del archivo cuenta.cpp

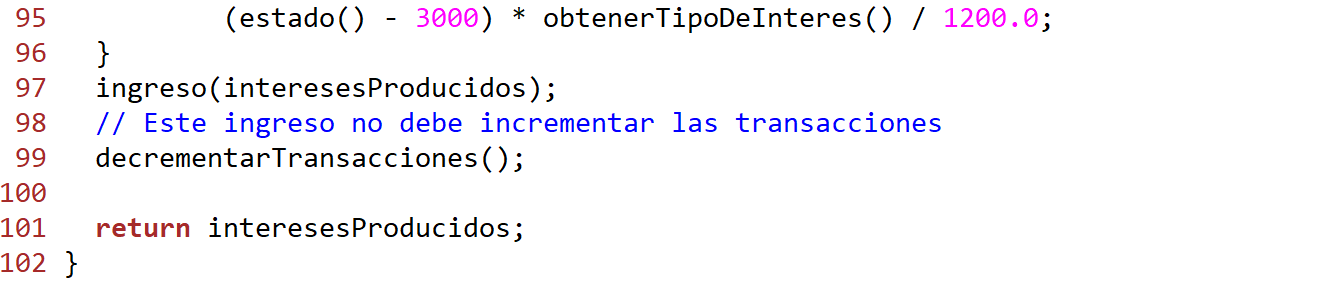
El método intereses de la clase CCuentaAhorro



Fragmento del archivo cuenta\_ahorro.cpp

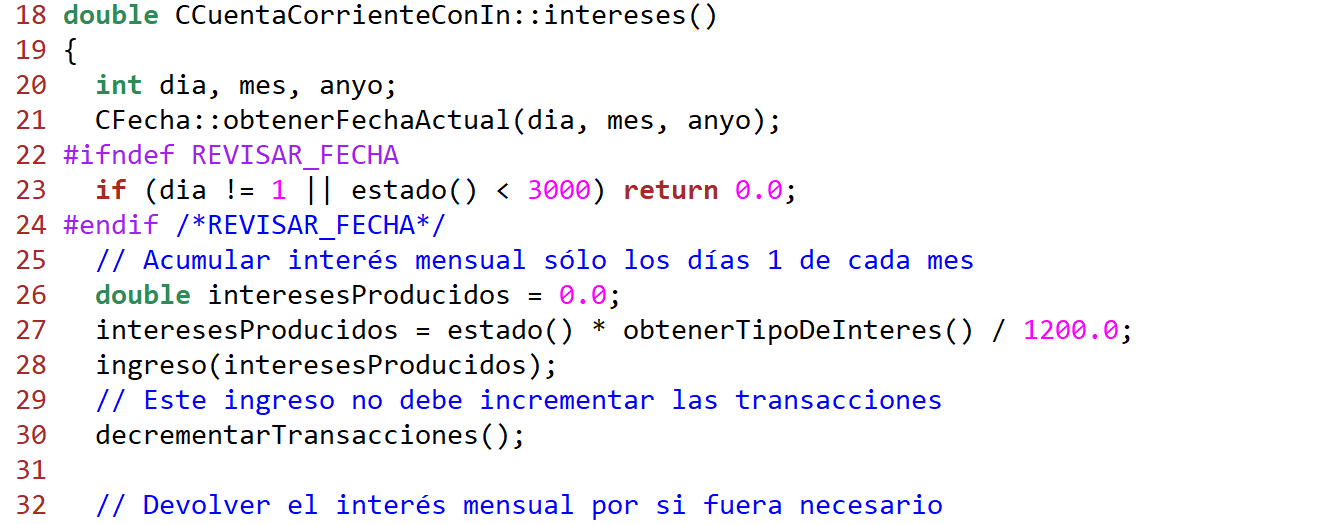
EL método intereses() de la clase CCuentaCorriente

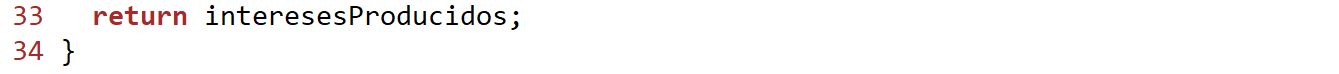




Fragmento del archivo cuenta\_corriente.cpp

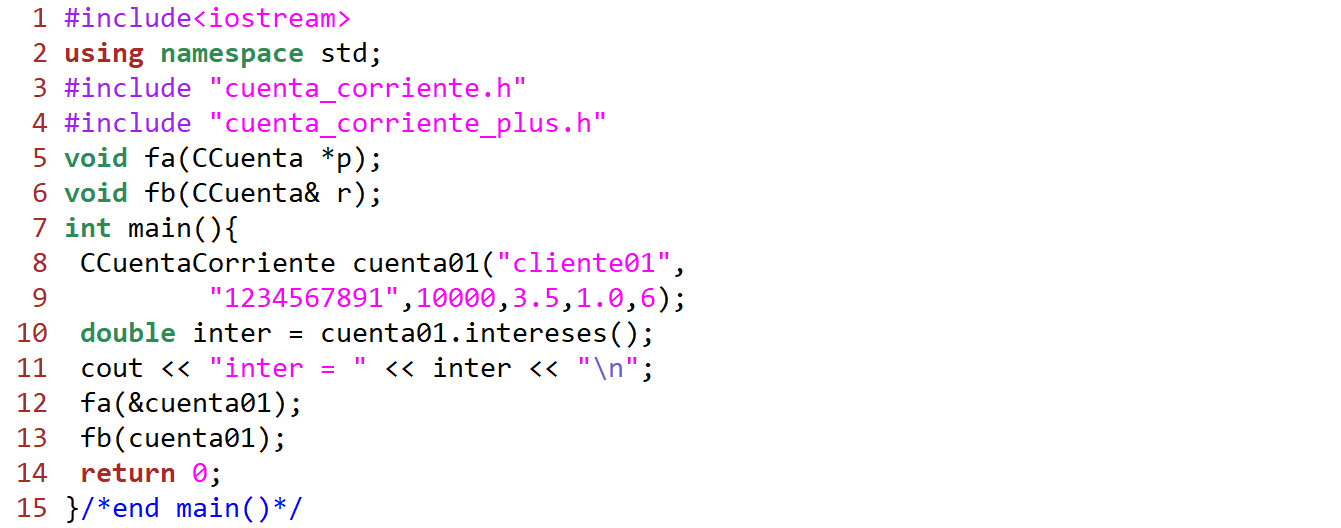
El método intereses() de la clase CCuentaCorrienteConIn

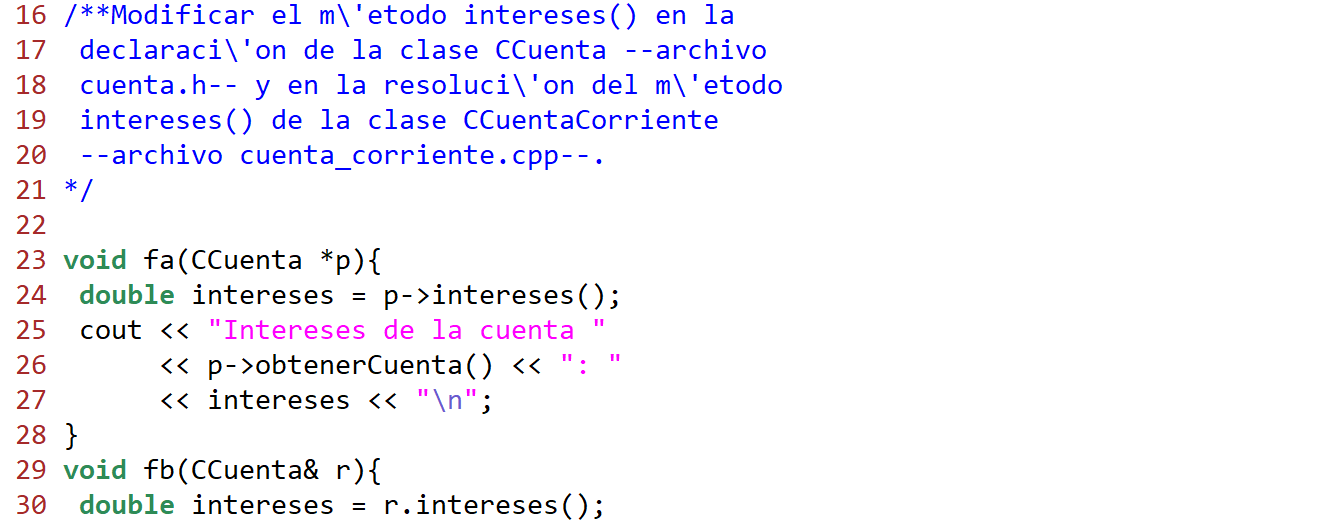


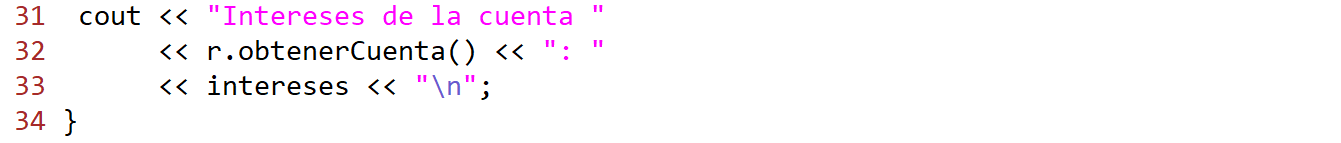


Fragmento del archivo cuenta\_corriente\_plus.cpp

Si ahora se ejecuta el código siguiente:







Archivo testMetodosVirtuales.cpp

El comportamiento del compilador es el esperado, aunque quizás no el deseado, ya que en ambos casos el método intereses() que se ejecuta pertenece a la clase CCuenta, que como podemos observar es el tipo de dato de \*p y de la referencia r, cuando quizás deseábamos que se ejecutase el método intereses() de la clase del objeto referenciado.

La solución al problema planteado es que sea el mismo sistema el que se encargue de la identificación de los objetos apuntados durante la ejecución, mecanismo que C++ proporciona por medio de los métodos virtuales.

Un **método virtual** es un miembro de una clase base que puede ser redefinido en cada una de las clases derivadas de ésta, y una vez redefinido puede ser accedido mediante un **apuntador** o un**a referencia** a la **clase base**, resolviéndose la llamada en función del tipo del objeto referenciado. Una clase con métodos virtuales se denomina ***tipo polimórfico***.

Un método se declara virtual escribiendo la palabra clave **virtual** al principio de la declaración del método en la clase donde aparece por primera vez. Las redefiniciones que realicemos de este método en las clases derivadas no necesitan incorporar en su declaración la palabra clave **virtual**, porque ya son declarados implícitamente métodos virtuales; hacerlo sería redundante.

*La redefinición de un método virtual en una clase derivada debe tener el mismo nombre, número y tipo de parámetros y tipo de valor retornado que en la clase base;* en otro caso, se producirá un error (hay una excepción que comentaremos más adelante al hablar de constructores virtuales). Además, una clase derivada puede contener sus propios métodos virtuales; esto es, métodos virtuales no heredados de sus clases base.

Ejemplo de un tipo polimórfico

En la clase CCuenta, se edita el archivo cuenta.h procediendo como se indica a continuación:



Una vez realizada la modificación anterior, si volvemos a ejecutar el ejemplo previo comprobaremos que:

p->intereses() llama al método CCuentaCorriente::intereses

r.intereses() llama al método CCuentaCorriente::intereses

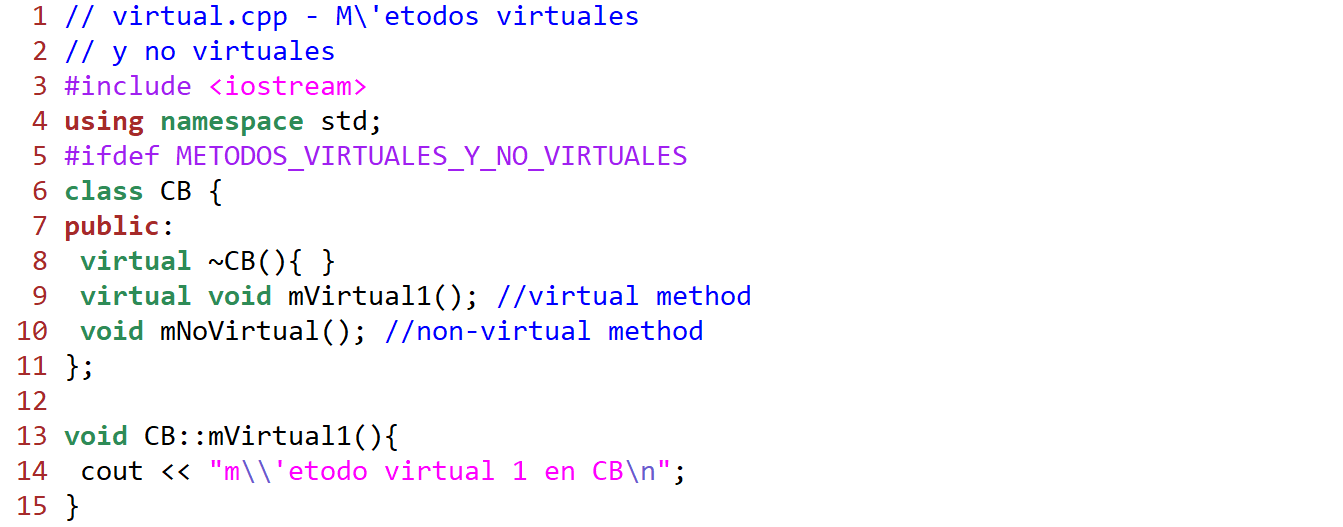
Por lo tanto, el método invocado pertenece, como deseábamos, a la misma clase que el objeto referenciado por p o por r (en el ejemplo, el objeto referenciado es cuenta01 de la clase CCuentaCorriente). El mecanismo virtual garantiza que el objeto será manipulado por los métodos de su clase.

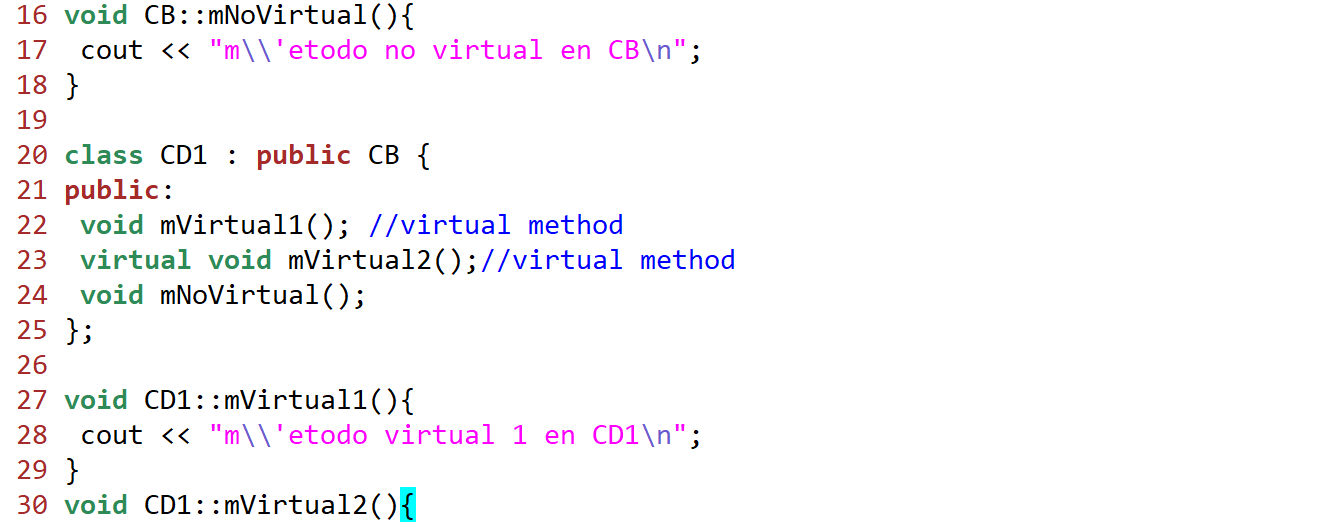
Si una clase derivada no provee una redefinición de un método declarado virtual en su clase base, una llamada al mismo hace que se ejecute el definido en su clase base. Por ejemplo, si CCuentaCorriente no redefiniera el método intereses, la llamada p->intereses() del ejemplo anterior invocaría al método intereses() de la clase CCuenta. Éste es el motivo de porqué un método virtual se declara en la clase base. Dicho de otra forma, cuando la clase derivada no redefine el método virtual de su clase base, hereda la implementación de la clase base, de manera que una llamada al mismo a través de un objeto de la clase derivada hace que se ejecute el método virtual de su clase base.

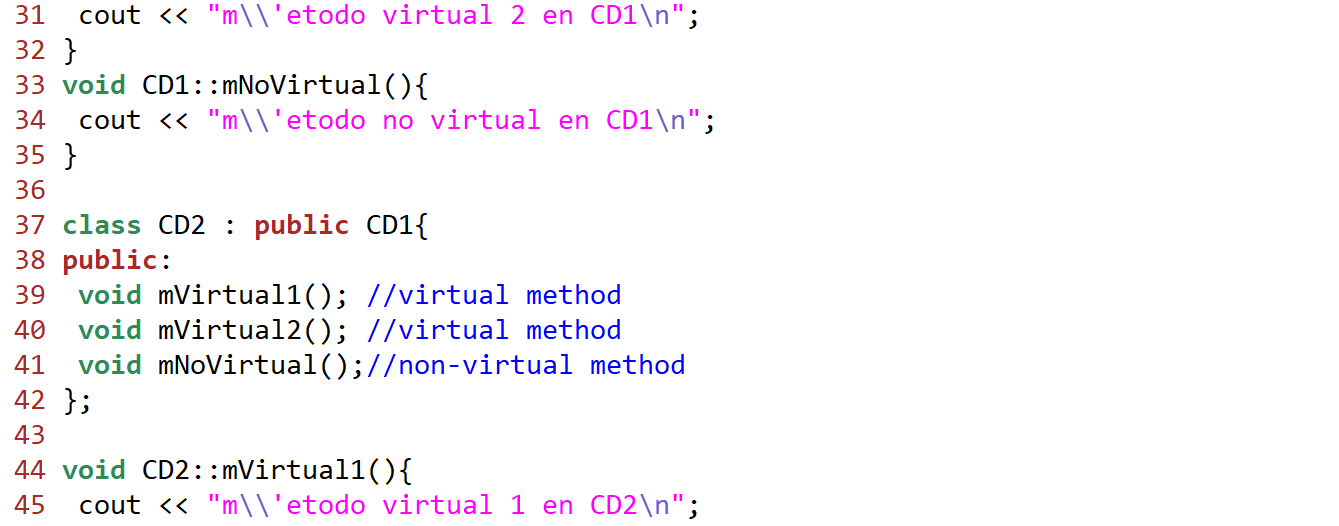
En resumen:

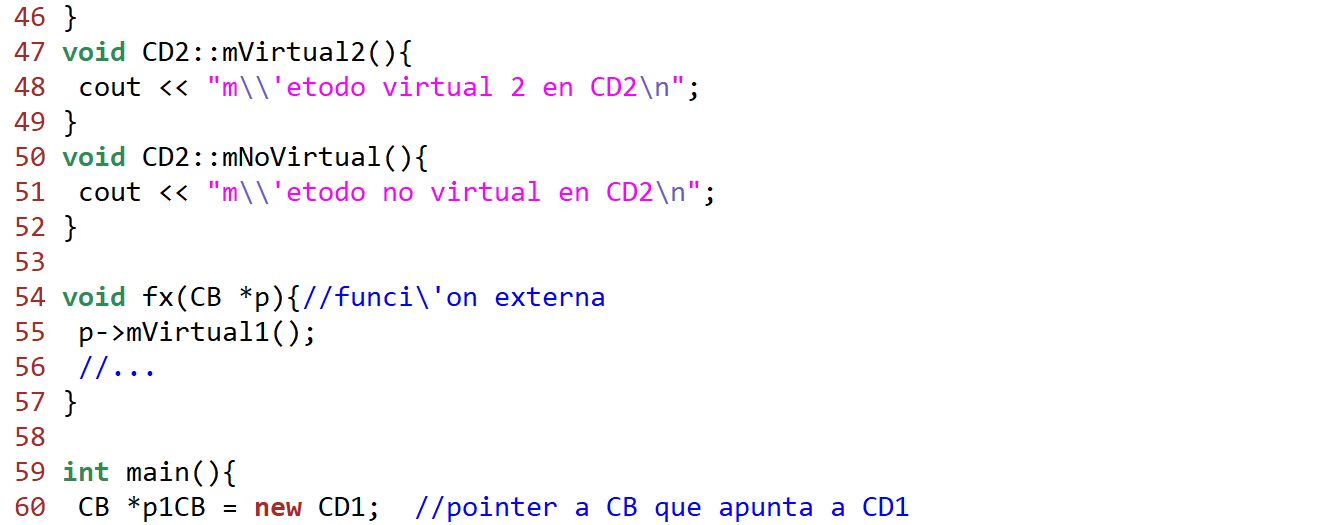
* Una llamada a un método virtual se resuelve siempre en función del tipo del objeto referenciado.
* Una llamada a un método normal (no virtual) se resuelve en función del tipo del apuntador o de la referencia.
* Una llamada a un método virtual específico exige utilizar el operador :: de resolución de alcance. Esta forma de llamar a un método virtual suprime el mecanismo *virtual*.

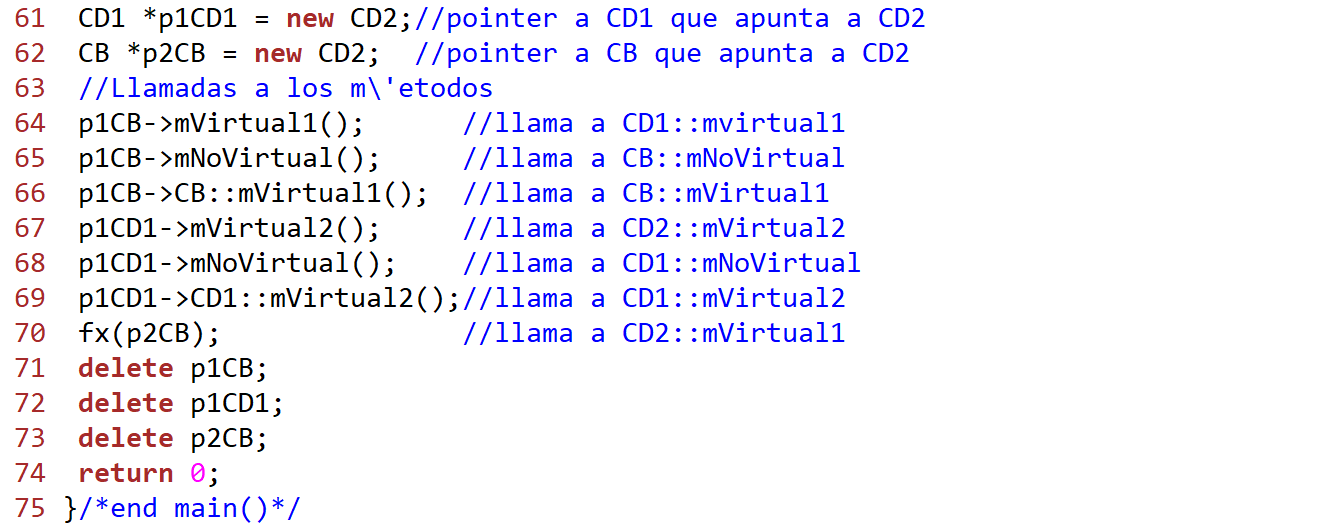
El siguiente ejemplo muestra de una forma práctica cómo se ejecutan las llamadas a métodos virtuales y no virtuales a través de apuntadores.

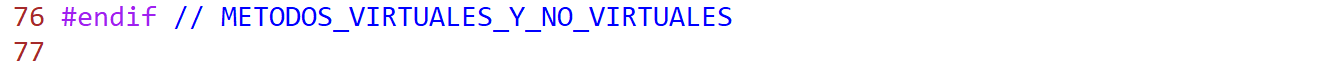












Archivo virtual.cpp

Observe cómo una llamada a un método virtual se resuelve en función del tipo del objeto apuntado, cómo una llamada a un método no virtual se resuelve en función del tipo del apuntador o de la referencia y cómo una llamada explícita, utilizando el operador :: de resolución de alcance, a un método virtual suprime el mecanismo virtual. Fíjese también que la clase CD1 define su propio método virtual, además de redefinir el método virtual heredado.

Una función externa o static no puede ser declarada **virtual**, ya que un método virtual solo es llamado para objetos de su clase. Sin embargo, un método virtual si puede ser declarado friend de otra clase.

# Constructores virtuales

Un método virtual tiene que invocarse para un objeto existente, pero un constructor no, ya que su función es construir el objeto; desde este punto de vista, no tiene sentido hablar de un constructor virtual. Un método virtual es invocado a través de un apuntador o una referencia, pero la forma del método que se invoca será una u otra en función del tipo del objeto referenciado. En cambio, un constructor es exclusivo de un tipo exacto de objetos, otra razón por la que no puede ser virtual. Entonces, la respuesta a la pregunta ¿pueden definirse constructores virtuales? es que C++ no admite constructores virtuales, pero resulta fácil simularlos. Supongamos una clase CB:

CB \*p = new CB; //Invoca al constructor CB

Esta operación podría también formar parte del cuerpo de un método que devuelve el objeto construido.

CB \*nuevo()

{

return new CB; // invoca al constructor CB

}

Si este método lo declaramos virtual en una clase base CB y lo redefinimos en una clase derivada CD para que devuelva un objeto CD, tendremos la posibilidad de crear un nuevo objeto sin conocer su tipo exacto. Por lo tanto, ***nuevo*** es un **método que simula a un constructor virtual**.

Análogamente, podemos escribir otro método virtual ***clonar*** que devuelva un duplicado del objeto para el cual es invocado, objeto que sabemos está referenciado por **this**. Tenemos, así, otro método que simula a un constructor copia virtual.

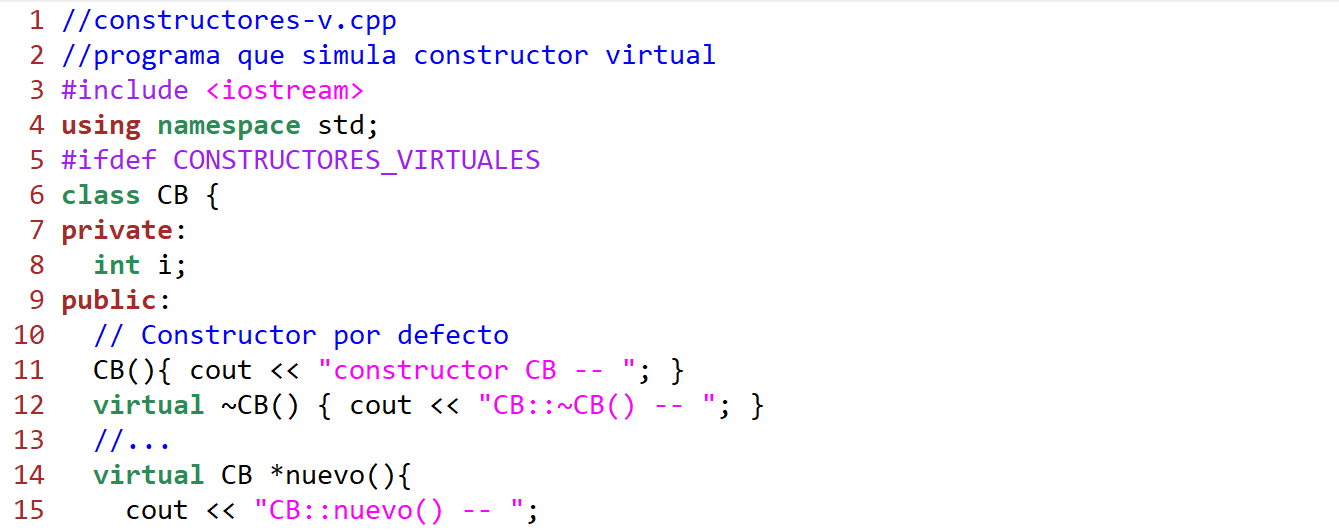
CB \*clonar()

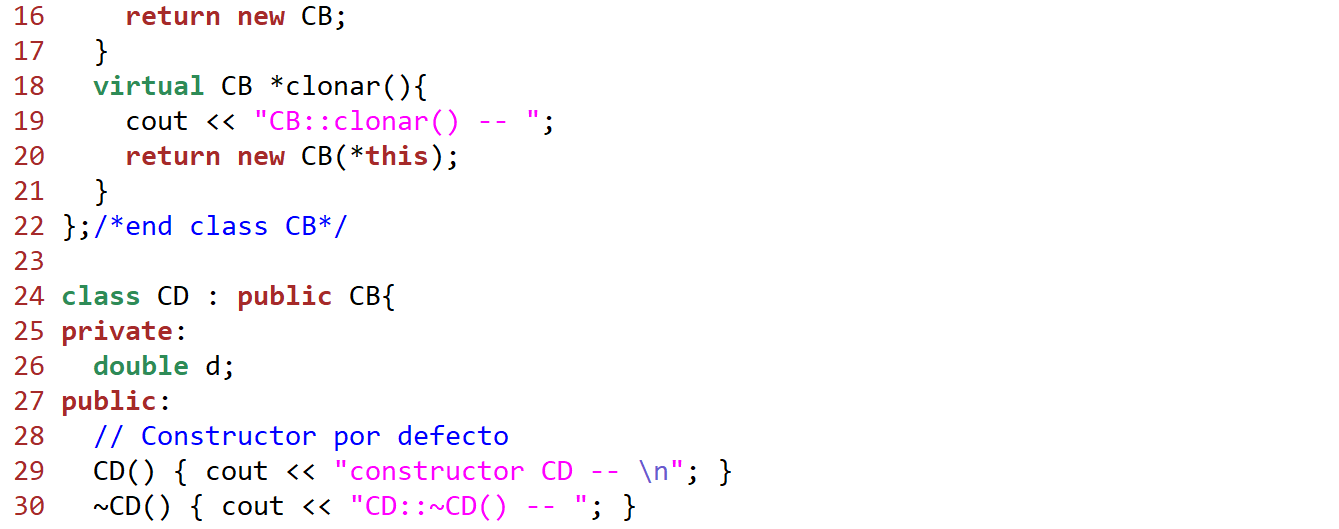
{

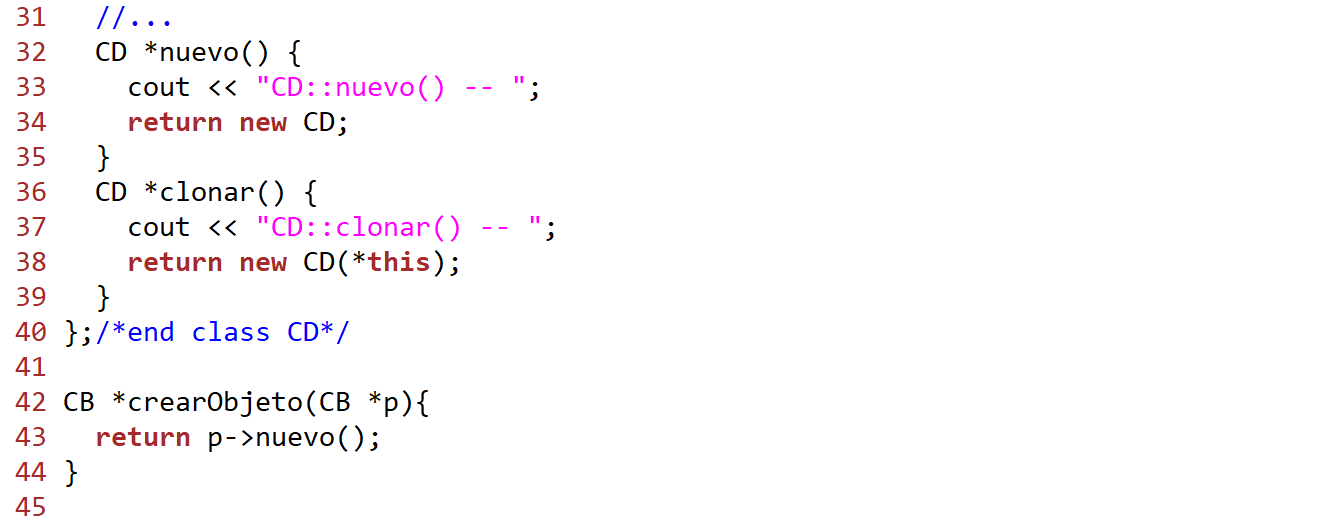
return new CB(\*this); // invoca al constructor copia ce CB

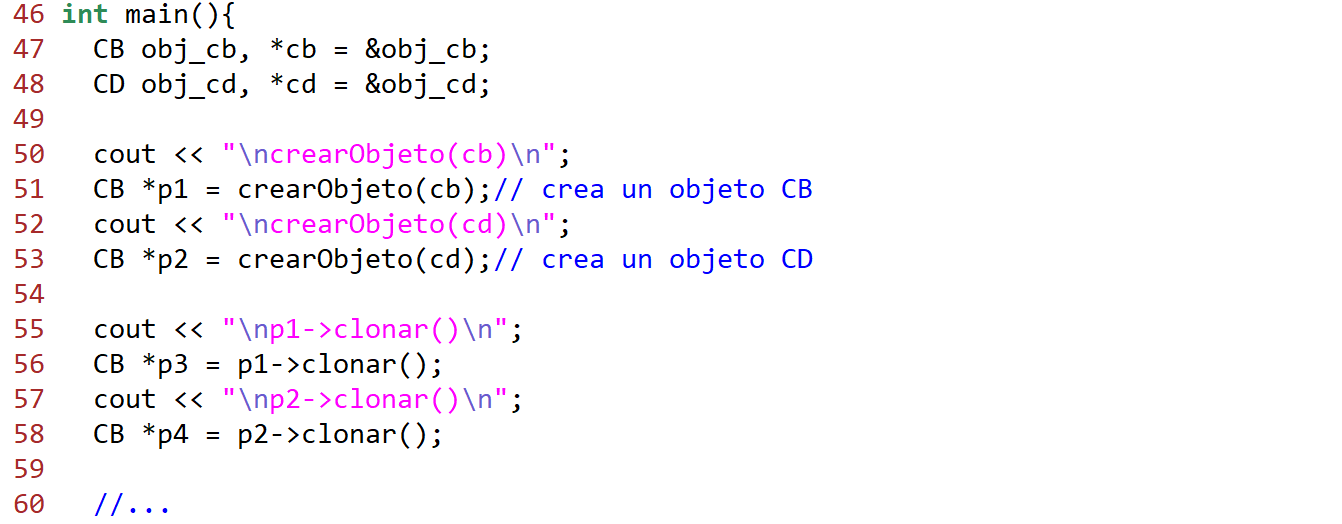
}

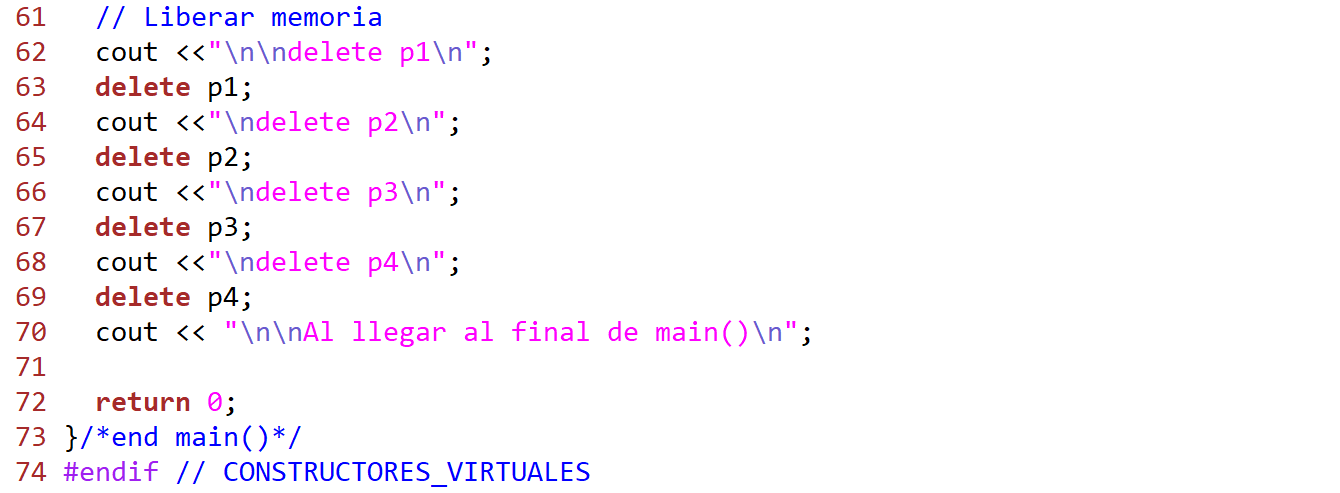
El programa que se muestra a continuación pone en práctica lo expuesto:











Archivo constructores-v.cpp

En el listado anterior, se logra el efecto de un constructor virtual de la clase CB mediante el uso de la función CB \*creaeObjeto(CB \*p), y se logra el efecto de un constructor copia virtual mediante el método virtual clonar().