# Una clase para representar números complejos

En esta sección se desarrolla una clase de números complejos. Se desarrolla esta clase de números complejos para mostrar el desarrollo de la misma. Cabe mencionar que C++ tiene un tipo de dato número complejo el cual está basado en plantillas.

Un número complejo tiene parte real y parte imaginaria. Una clase de números complejos deberá tener atributos que representen esas cantidades. El constructor por defecto debe establecer ambas partes (real e imaginaria) a cero. También sería deseable tener un constructor que permita establecer el número complejo z=x+jy, donde x y y son variables de punto flotante de doble precisión, usando sentencias de la forma

double x = 4.0;

double y = -3.0;

ComplexNumber z(x,y);

Además, podríamos también incluir miembros de la clase que sean métodos que calculen el módulo y el argumento de este número complejo. Otro método podría usarse para elevar el número complejo a una potencia especificada.

Sobrecarga de operadores

Si hemos declarado a, b, c y d como variables enteras entonces podríamos fácilmente relacionar esas variables a través de sentencias como

int a,b,c,d;

a=b;

c=-a;

d=a+b;

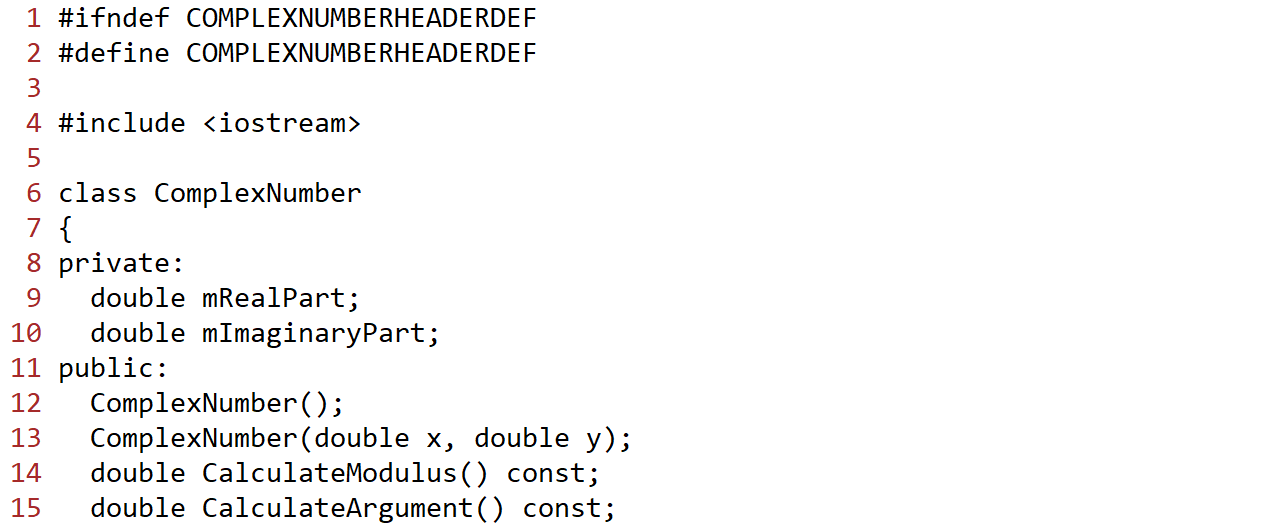
Nos gustaría también escribir sentencias como estas si a, b, c y d fueran números complejos en lugar de enteros. Antes de que podamos hacer esto, necesitamos definir: (i) que significa el operador de asignación para número complejos; (ii) qué significa el operador unario menos ---esto es, qué significa la expresión “-a” si a es un número complejo; y (iii) qué significa el operador de adición binaria ---esto es, qué significa a+b para a y b números complejos. Definir esos operadores para clases se conoce como *sobrecarga de operadores*.

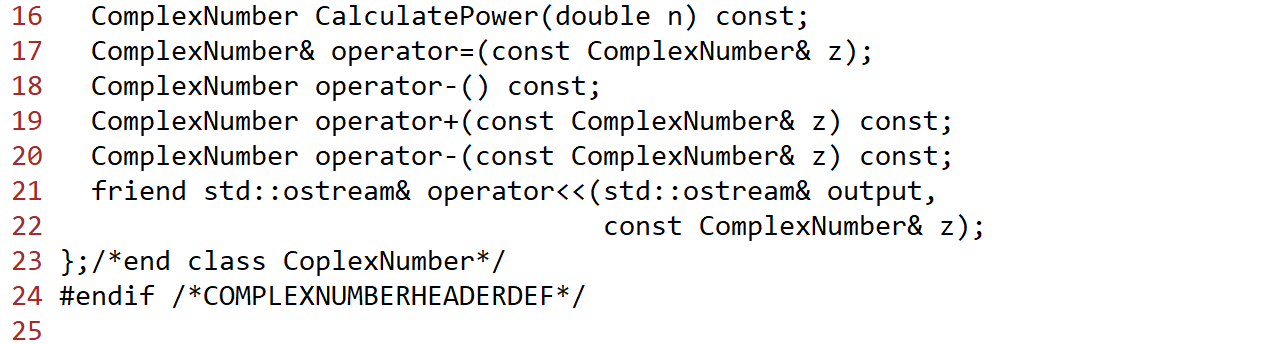
La clase de números complejos

A la luz de la discusión presentada, escribiremos una clase de números complejos con los siguientes miembros.

* Una variable de punto flotante de doble precisión mRealPart que contiene la parte real del número complejo.
* Una variable de punto flotante de doble precisión mImaginaryPart que contiene la parte imaginaria del número complejo.
* Un constructor por defecto sobrecrito ComplexNumber() que inicializa la parte real y la parte imaginaria a cero.
* Un constructor ComplexNumber(double x,double y) que inicializa la parte real a x y la parte imaginaria a y.
* Un método CalculateModulus() que devuelve una variable de punto flotante de doble precisión que contiene el módulo (o magnitud) del número complejo.
* Un método calculateArgument() que devuelve una variable de punto flotante de doble precisión que contiene el argumento (o fase) del número complejo.
* Un método CalculatePower(double n) que devuelve el número complejo calculado cuando se eleva el número complejo original a la potencia n.
* Sobrecarga del operador de asignación.
* Sobrecarga del operador unario de substracción.
* Sobrecarga de los operadores binarios de adición y de substracción.
* Sobrecarga del operador de inserción de flujo de salida (<<) lo cual da control del formato de salida para números complejos.

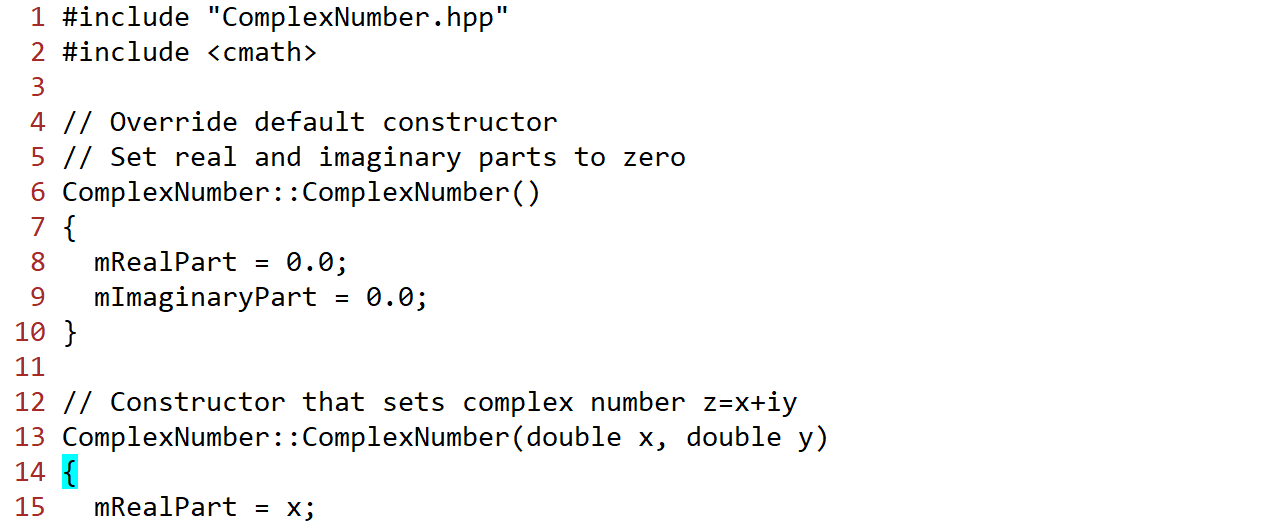
Un archivo de cabecera adecuado para esta clase se muestra abajo. Este debe ser guardado como ComplexNumber.hpp. Hemos hecho los datos asociados con cada número complejo ---esto es, la parte real y la parte imaginaria--- miembros privados de esta clase para evitar modificaciones inadvertidas de esos miembros. Esos miembros, por supuesto, ser accedidos por los métodos de la clase.

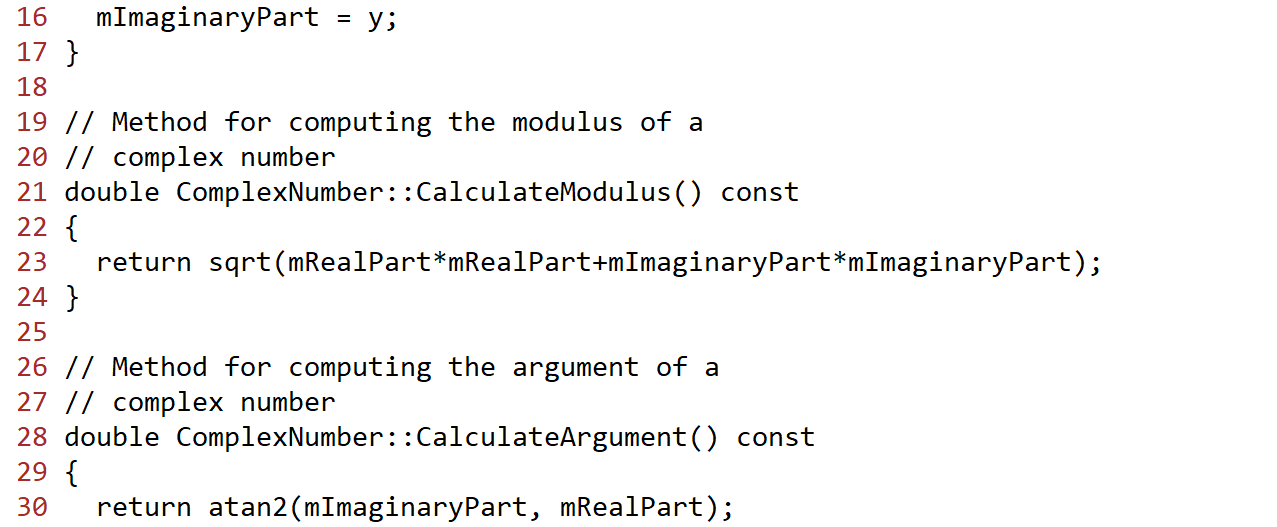


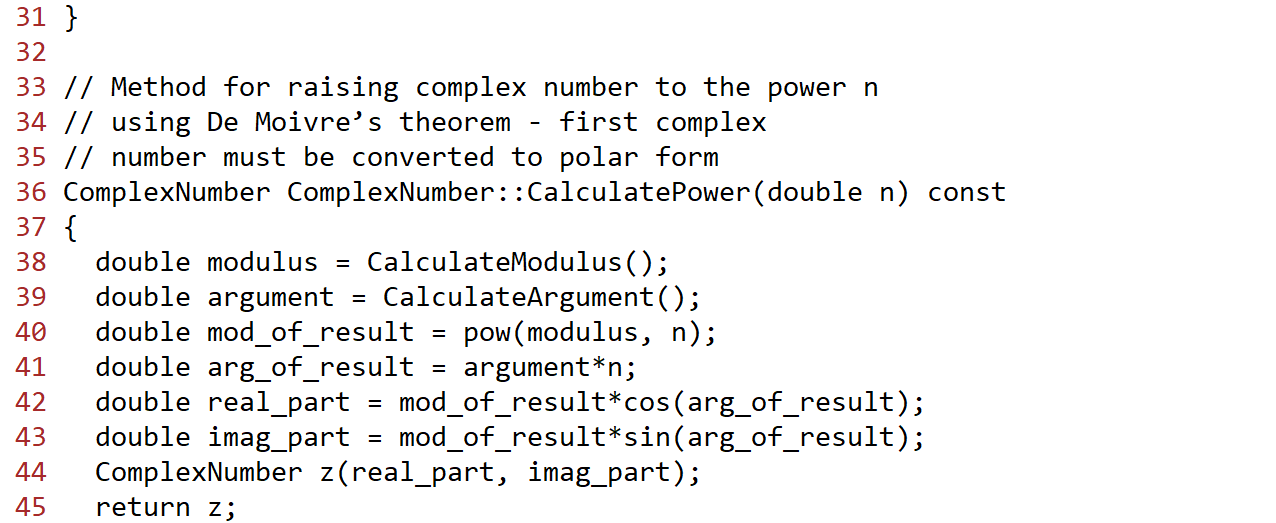


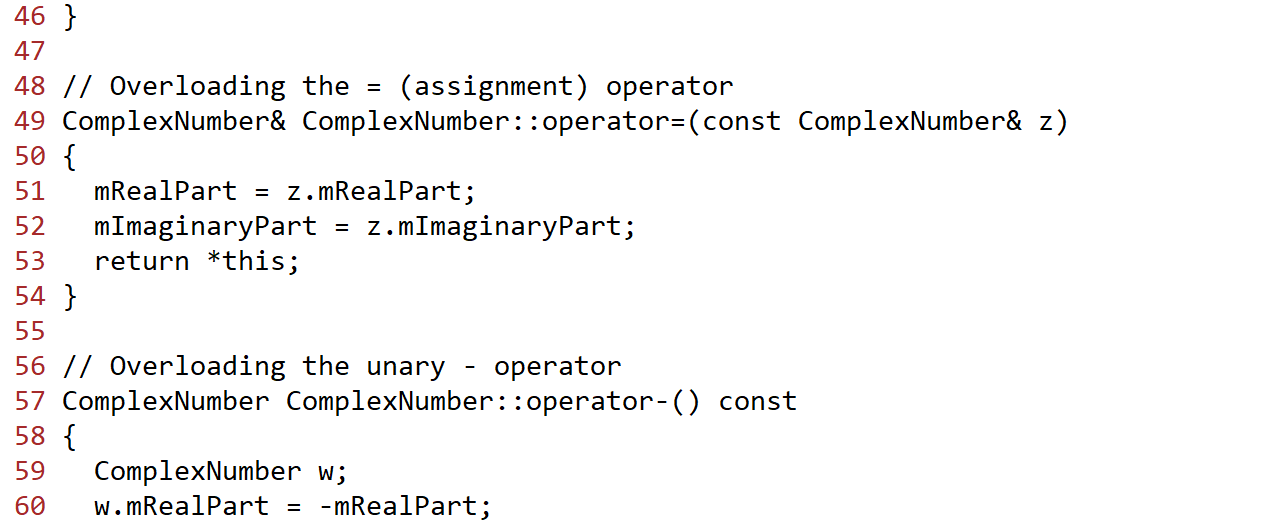
Archivo ComplexNumber.hpp

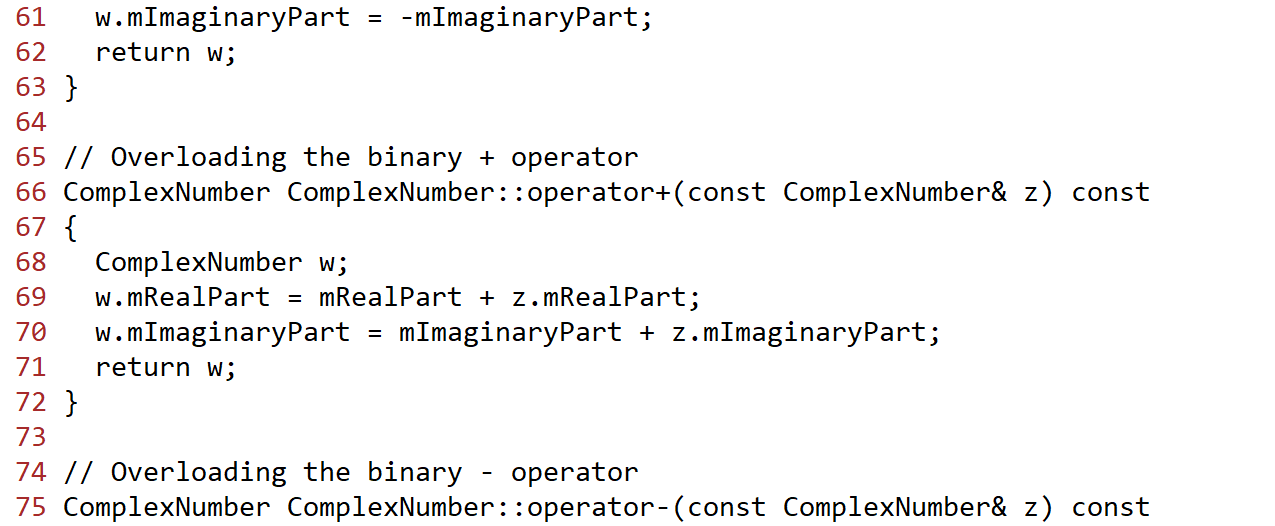
A continuación, se muestra el código del archivo ComplexNumber.cpp

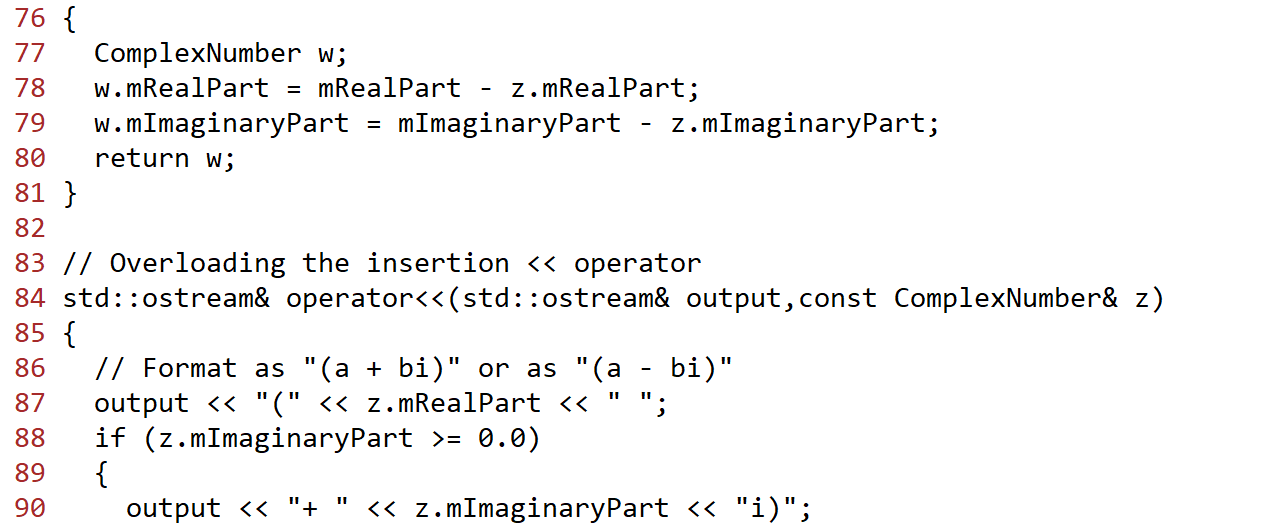


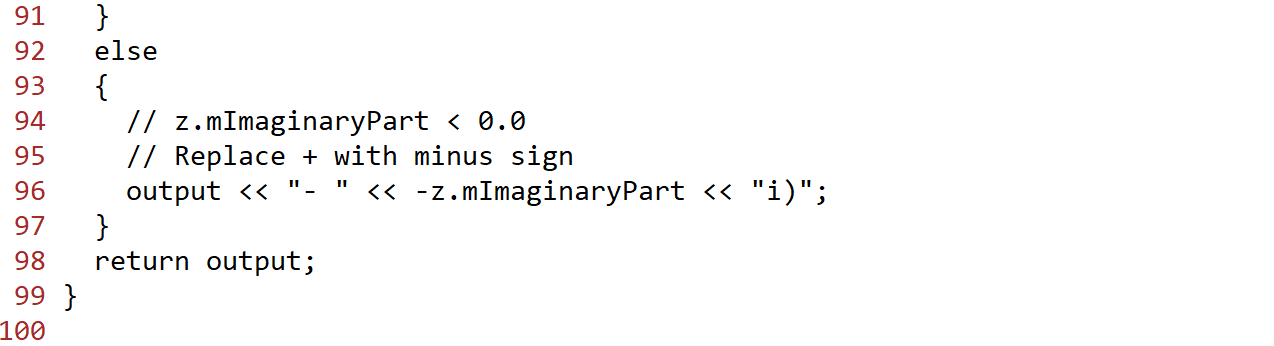








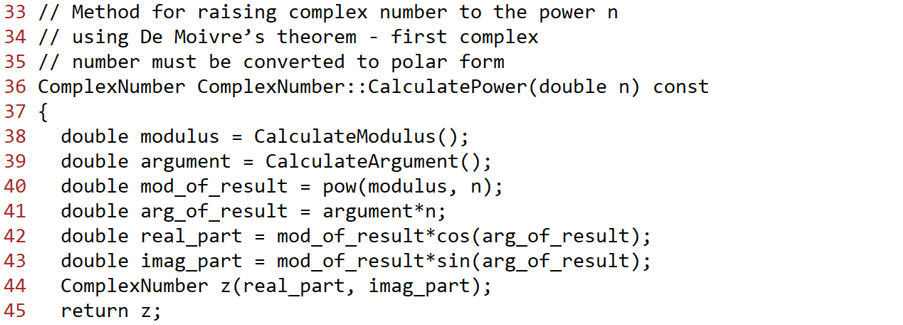




Archivo ComplexNumber.cpp

Algunos comentarios sobre el código en el archivo ComplexNumber.cpp

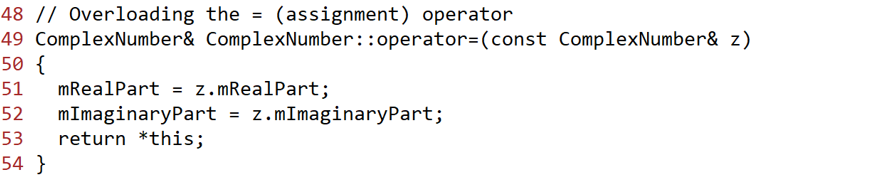
En el método CalculatePower especificamos que la variable devuelta es de tipo





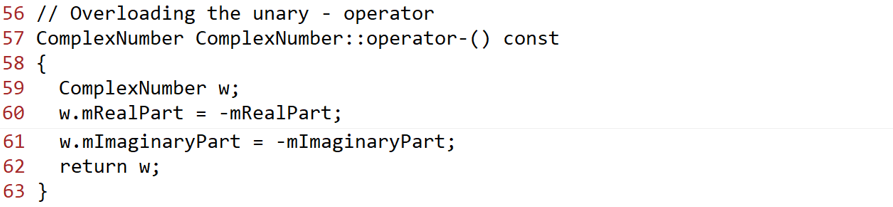
ComplexNumber: esto es, los métodos pueden ser usados para devolver una instancia de una clase de la misma forma como si devolviera una variable de tipo más simple tal como un double. Este método también requiere como entrada el exponente al cual se eleva el número complejo: este es especificado por el argumento “double n”. Dentro del método, las primeras dos líneas del código calculan el módulo y el argumento del número original usando los dos miembros de clase CalculateModulus y CalculateArgument ---est demuestra cómo llamar esos métodos desde el interior de la clase. Las siguientes dos líneas realizan los cálculos requeridos sobre el módulo y el argumento del número complejo para elevarlo a la potencia n. Habiendo establecido la parte real e imaginaria del número complejo resultante, este número complejo es el que se devuelve.

En la sobrecarga del operador de asignación, nótese que el argumento para el



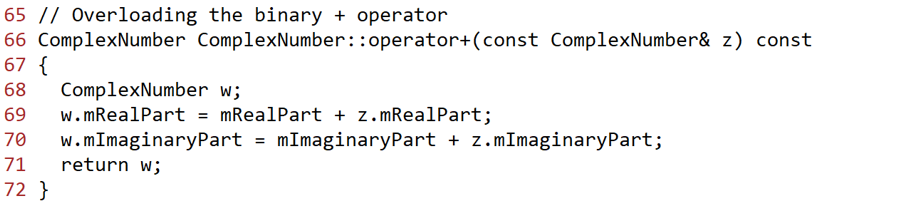
operador de asignación es una referencia a otra instancia de la clase en lugar de un objeto. Esto es porque, por defecto, todos los argumentos de los métodos son llamados por valor (llamados por copia) haciendo necesario el trabajo adicional del uso del constructor de copia al hacer la asignación. El uso de la palabra reservada const garantiza que el operador de asignación no alterará el contenido del objeto argumento z. El resto del método para asignación usa una entidad llamada this la cual no parece haber sido declarada. Para nuestros propósitos, solo necesitamos saber que this es un apuntador al número complejo que es devuelto: es el contenido de this lo que es devuelto.

El operador de substracción unario también está sobrecargado: (i) el tipo de retorno



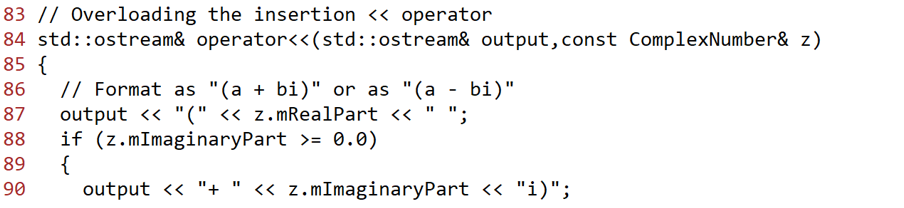
es un ComplexNumber; (ii) el método es un miembro de la clase ComplexNumber; (iii) define el operador “-”; (iv) la función no requiere argumentos de entrada (como se especifica por los paréntesis vacíos); y (v) el número complejo original se deja sin cambio (a través del uso de const). Se declara una instancia de la clase, llamada w, y las partes real e imaginaria de w se hacen igual a los inversos aditivos de las partes real e imaginaria del número complejo original. Finalmente, se devuelve el número complejo w.

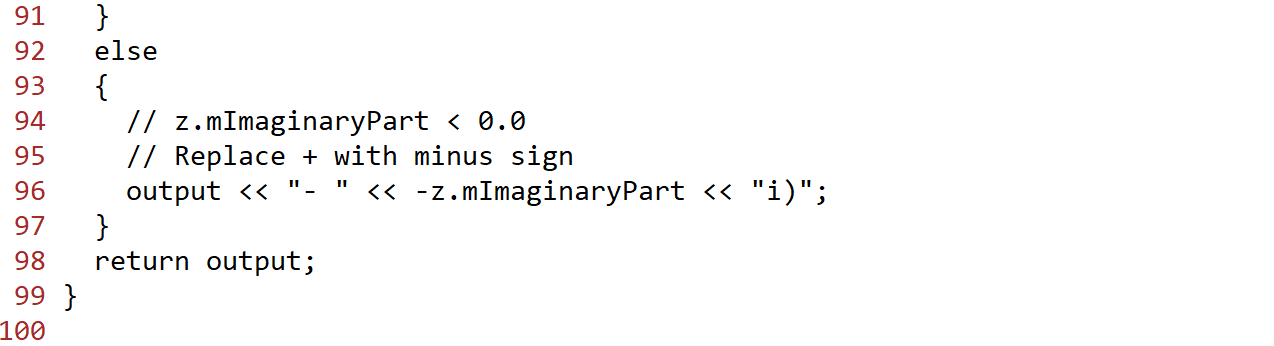
El operador de adición binario se define como sigue:



Comenzamos como es usual especificando el tipo de retorno, el nombre de la clase de la que la función es miembro, el operador y el argumento de entrada. Hay solamente un argumento de entrada el cual es el de la derecha del operador + ---en cierto sentido, la clase misma (o un objeto implícito de la clase misma) es el argumento u operando izquierdo. Declaramos una instancia de un número complejo, realizamos la adición requerida, y devolvemos el resultado de esta suma. Con una función similar se sobrecarga el operador binario de substracción.

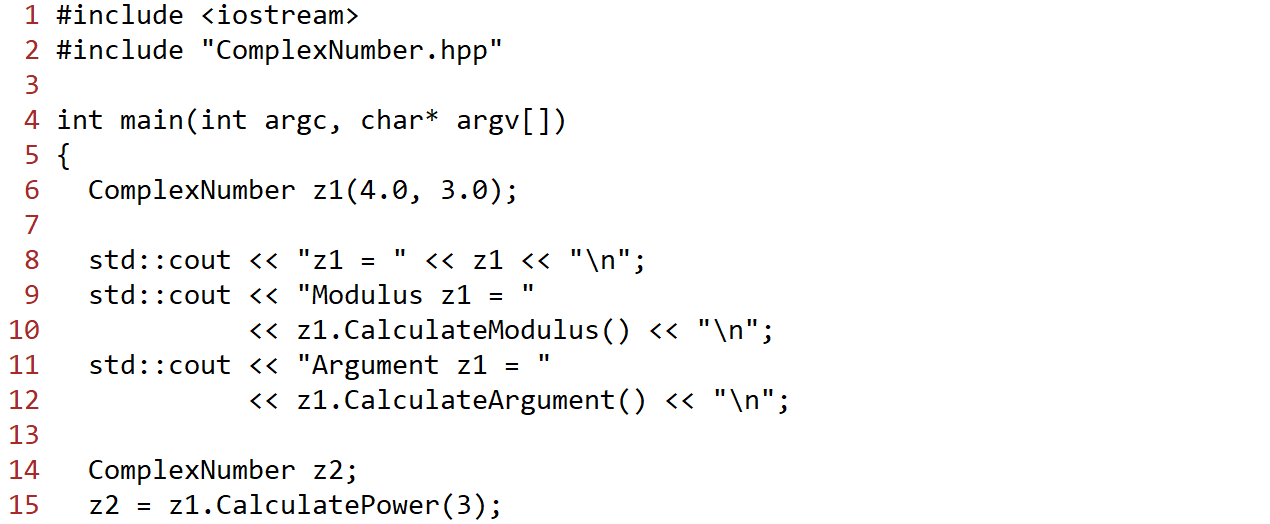
La sobrecarga del operador de inserción a flujo de salida (<<). La sintaxis aquí es diferente:

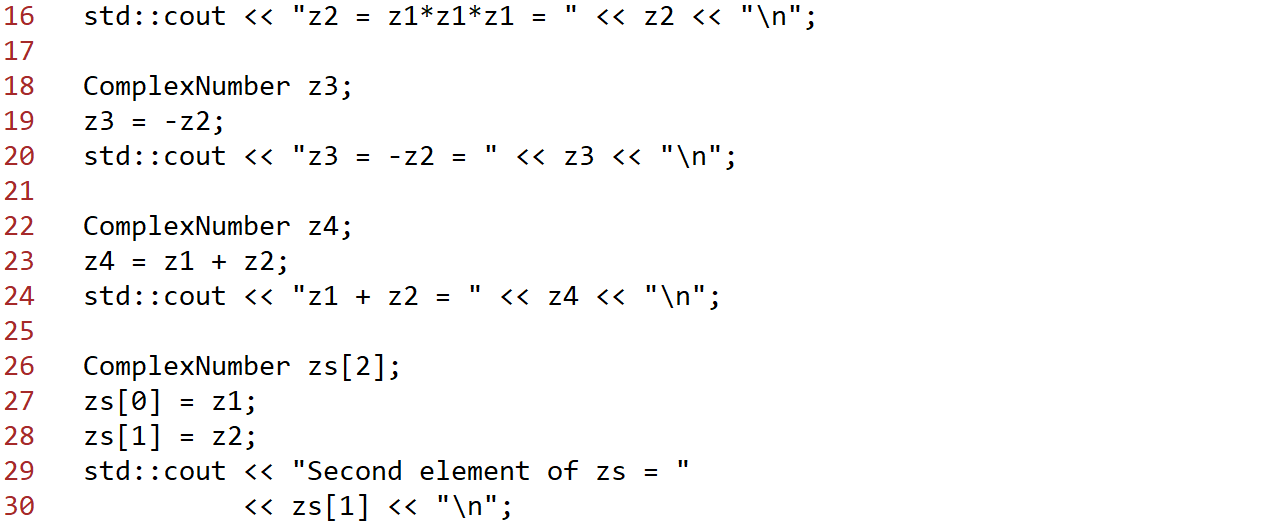




El operador no es un método miembro de la clase, sino una función externa. Este operador usa la palabra clave friend. Usando la palabra clave friend para el operador << en el archivo de cabecera para la clase CompexNumber, estamos diciendo al compilador que, aunque este operador no es un miembro de la clase, este operador puede acceder a todos los miembros de la clase ---incluyendo los miembros privados. En el código mostrado arriba, de la sobrecarga del operador <<, vemos que no lo hacemos un miembro de la clase porque en este caso, no usamos ComplexNumber::. La función que define este operador toma un flujo de salida (tal como std::cout o un flujo de salida a un archivo) e inserta caracteres en él usando el número complejo z.

Ahora demostramos el uso de la clase ComplexNumbers en el siguiente código. Recordemos que cuando se llama a métodos miembro que no requieren argumentos, aun así, se le deben agregar paréntesis vacíos, por ejemplo, z1.CalculateModulus() en el código siguiente. Nótese que podemos declarar un arreglo de números complejos: en el ejemplo siguiente se declara un arreglo de números complejos con dos entradas. El primer elemento de este arreglo lo hacemos igual a z1, y el segundo elemento de este arreglo lo hacemos igual a z2. También en el ejemplo, mostramos como usar una función amiga (friend) de una entrada de un arreglo, a través de la impresión a la pantalla del número complejo que es la segunda entrada del arreglo de números complejos.







Archivo TestComplexNumber.cpp