# Use of explicit keyword in C++

Predict the output of following C++ program.

|  |
| --- |
| #include <iostream>    using namespace std;    class Complex  {  private:      double real;      double imag;    public:      // Default constructor      Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}        // A method to compare two Complex numbers      bool operator == (Complex rhs) {         return (real == rhs.real && imag == rhs.imag)? true : false;      }  };    int main()  {      // a Complex object      Complex com1(3.0, 0.0);        if (com1 == 3.0)         cout << "Same";      else         cout << "Not Same";       return 0;  } |

Output: The program compiles fine and produces following output.

Same

As discussed in [this GFact](https://www.geeksforgeeks.org/archives/10890), in C++, if a class has a constructor which can be called with a single argument, then this constructor becomes conversion constructor because such a constructor allows conversion of the single argument to the class being constructed.  
We can avoid such implicit conversions as these may lead to unexpected results. We can make the constructor explicit with the help of [*explicit keyword*](http://msdn.microsoft.com/en-us/library/h1y7x448%28v=vs.80%29.aspx). For example, if we try the following program that uses explicit keyword with constructor, we get compilation error.

|  |
| --- |
| #include <iostream>    using namespace std;    class Complex  {  private:      double real;      double imag;    public:      // Default constructor      explicit Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}        // A method to compare two Complex numbers      bool operator== (Complex rhs) {         return (real == rhs.real && imag == rhs.imag)? true : false;      }  };    int main()  {      // a Complex object      Complex com1(3.0, 0.0);        if (com1 == 3.0)         cout << "Same";      else         cout << "Not Same";       return 0;  } |

Output: Compiler Error

no match for 'operator==' in 'com1 == 3.0e+0'

We can still typecast the double values to Complex, but now we have to explicitly typecast it. For example, the following program works fine.

|  |
| --- |
| #include <iostream>    using namespace std;    class Complex  {  private:      double real;      double imag;    public:      // Default constructor      explicit Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}        // A method to compare two Complex numbers      bool operator== (Complex rhs) {         return (real == rhs.real && imag == rhs.imag)? true : false;      }  };    int main()  {      // a Complex object      Complex com1(3.0, 0.0);        if (com1 == (Complex)3.0)         cout << "Same";      else         cout << "Not Same";       return 0;  } |

Output: The program compiles fine and produces following output.

Same