**Explicit Keyword in C++** is used to mark constructors to not implicitly convert types in C++. It is optional for constructors that take exactly one argument and work on constructors(with a single argument) since those are the only constructors that can be used in typecasting.

|  |
| --- |
| **//C++ program to illustrate default constructor without 'explicit' keyword**  #include <iostream>  using namespace std;    class Complex {  private:      double real;      double imag;    public:        // Parameterized 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);      }  };    // Driver Code  int main()  {      // a Complex object      Complex com1(3.0, 0.0);        if (com1 == 3.0)          cout << "Same";      else          cout << "Not Same";      return 0;  } |

**Output**

Same

|  |
| --- |
| **// C++ program to illustrate default constructor with 'explicit' keyword**  #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);      }  };    // Driver Code  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 receive an error here because to avoid any unexpected errors we have made our constructor an explicit constructor and 3.0 won’t be converted to Complex by our constructor on its own.

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

|  |
| --- |
| **// C++ program to illustrate default constructor with 'explicit' keyword**  #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);      }  };    // Driver Code  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**

Same

***Note:*** *The explicit specifier can be used with a constant expression. However, if that constant expression evaluates to true, then only the function is explicit.*