GeeksforGeeks

Related Articles

Use of explicit keyword in C++

Difficulty Level : Medium ● Last Updated : 28 Sep, 2018

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, 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. 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
```

Also see Advanced C++ | Conversion Operators

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Want to learn from the best curated videos and practice problems, check out the C++ Foundation Course for Basic to Advanced C++ and C++ STL Course for foundation plus STL. To complete your preparation from learning a language to DS Algo and many more, please refer **Complete Interview Preparation Course**.