# 12+ Operator Overloading

## Conversion Operators

Hu Sikang
skhu@163.com

# Example: ostream <<

**The *operator <<* can be defined as a binary operator. In general, the *operator <<* is defined as a friend member function of class and has two arguments: one is the reference of ostream, the other is an object.**

```
class complex {
public:
    complex(double x = 0, double y = 0)
    {  re = x;     im = y;  }
    void Display( )
     { cout << re << "+" << im << "i" << endl;  }
private:
    double re, im;
};
```

```
complex obj(10, 20);
cout << obj << endl;
```

# Example: ostream <<

```
#include <iostream>
using namespace std;
class complex {
public:
    complex(double x = 0, double y = 0)
    {   re = x;     im = y;   }
    friend ostream& operator <<(ostream& os, const complex& a);
private:
    double re, im;
};
ostream& operator <<(ostream& os, const complex& a)
{
    os << a.re << " + " << a.im << "i" << endl;
    return os;
}
```

```
void main()
{
        complex obj(10, 20);
        cout << obj << endl;
}
```

# Conversion Operators

**What is the conversion operators?**

```
complex obj(10, 0);
double x = 2.1;
x = obj;      // conversion operators
```

**So if we specify:**

**[1] an implicit conversion from a user-defined type to a basic type, or**

**[2] a conversion from a new class to a previously defined class,**

**we can use *conversion operators*.**

# Conversion Operators

```
#include <iostream>
using namespace std;
class Rational {
public:
   Rational(double x = 0, double y = 1)
   {
      Numerator = x;
      Denominator = y;
   }
   operator double( ) {
         return Numerator / Denominator;
    }
private:
      double Numerator, Denominator;
};
```

```
void main()
 {
     Rational r(100, 200);
     double d = r;
     cout << d << endl;
 }
```

# Example: vector

```
#include <iostream>
using namespace std;
class vector {
public:
   vector(int s)  { v = new int[s];   capacity = s;    size = 0;  }
   ~vector()       {  if (v != nullptr)    delete[] v;    }
   int& operator [ ](int i)     {    return v[i];    }
private:
      int *v;
      int capacity;  // number of elements' storage
      int size;          // number of current elements
};
```

```
void main()
{
      vector vec(5);
      vec[2] = 12;
      cout << vec[2] << endl;
}
```

If the code is written as follow, how to modify *Vector:*

```
if (vec)   // Is the "vec" empty?
    cout << " Some elements in vec." << endl;
else
    cout << " No element in vec." << endl;
```

# Conversion Operators

```
#include <iostream>
using namespace std;
class A {    };
class B {
public:
        operator A( )  {
                cout <<  " Convert B to A!"  << endl;
                return A( );  // Return an anonymous object of class A
        }
};
```

```
void main()
{
        B   b;
        A   a;
        a = b;   //convert
}
```