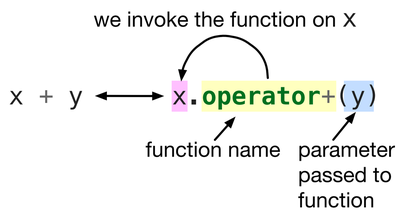



```
class Complex {
    private:
        double real;
        double img;
    public:
        Complex() {
            real = 0; img = 0;
        }
        Complex(double r, double i) {
            real = r; img = i;
        }
```

return type *function name* *input type* *object name (is `y` passed to `rhs`)*

```
        Complex operator+(Complex rhs) {
            Complex temp;
            temp.real = real + rhs.real;
            temp.img = img + rhs.img;
            return temp;
        }
};
```

*real of object on which **operator+** is invoked on (or x)*

*real of rhs (or y) passed to **operator+***

return by value

*Pass by reference to be **memory efficient***

```
Complex operator+(const Complex& rhs) const {
    Complex temp;
    temp.real = real + rhs.real;
    temp.img = img + rhs.img;
    return temp;
}
```

*if **rhs.real = 0;** is written in the function, you'll get a compile-time error!*

*if **real = 0;** is written in the function, you'll get a compile-time error!*

```
z = x + y;
```

`x.operator+(y)`

`z.operator=(x+y)`

```
z = y = x;
```

`y.operator=(x)` *returns original* `y`

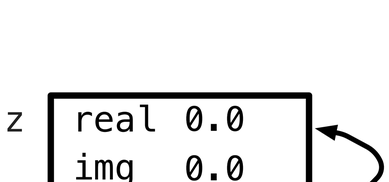
`z.operator=(y)` *returns original* `z`

```
class Complex {
    private:
        double real;
        double img;
    public:
        Complex() {
            real = 0; img = 0;
        }
        Complex(double r, double i) {
            real = r; img = i;
        }

        Complex& operator=(const Complex& rhs) {
            real = rhs.real;
            img = rhs.img;
            return *this;
        }
};
```

return by reference

*didn't add **const** as we want to change `real` and `img`*




```
cout << "z is " << z << endl;
```

returns original `cout`

prints z is

```
cout << z << endl;
```

returns original `cout`

prints real and img of z

```
cout << endl;
```

returns original `cout`

prints newline