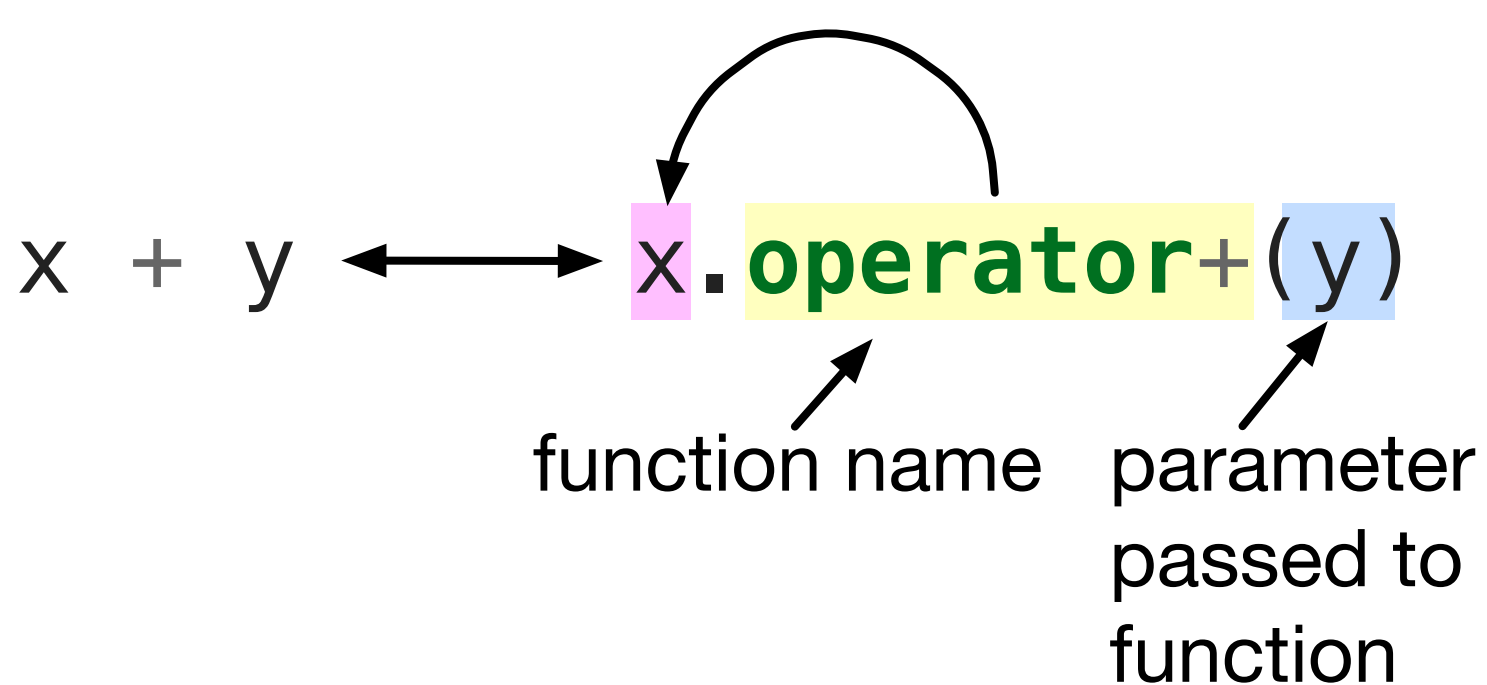


we invoke the function on X



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

The diagram shows the implementation of the `operator+` function for the `Complex` class. The function signature is `Complex operator+(Complex rhs)`. Annotations include: "return type" pointing to `Complex`, "function name" pointing to `operator+`, "input type" pointing to `Complex`, and "object name (is y passed to rhs)" pointing to `rhs`. Inside the function, `Complex temp;` is declared. The text "real of object on which operator+ is invoked on (or X)" points to `real` in `temp.real`. The text "real of rhs (or y) passed to operator+" points to `rhs.real`. The code `temp.real = real + rhs.real;` and `temp.img = img + rhs.img;` are shown. A purple arrow labeled "return by value" points from `return temp;` back to the `Complex` return type in the function signature. The function ends with `}` and the class definition ends with `};`.

Pass by reference to be **memory efficient**

```
Complex operator+(const Complex& rhs) const {  
    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!
```

```
    Complex temp;  
    temp.real = real + rhs.real;  
    temp.img = img + rhs.img;  
    return temp;  
}
```