SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220323<A|D>

Replace <A|D> with this section's letter

Operator Overloading and Inheritance Basics

CS 2124: Object Oriented Programming
Darryl Reeves, Ph.D.

Agenda

- Finish Operator Overloading Problem
- Background on Inheritance
- Inheritance Basics

In-class problem

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}

    // function for incrementing complex numbers (pre)
    Complex& operator++() {
       ++real;
       return *this;
    }

    // function for incrementing complex numbers (post)
    ---

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

now available: ++comp;

equivalent to: comp.operator++()

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    ---

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    ___ operator++(___) { }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    _6_ operator++(___) { }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

TurningPoint

SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220323<A|D>

Replace <A|D> with this section's letter

Which return type replaces blank #6 for the post-increment version of op++?

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    _6_ operator++(___) { }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(___) { }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

$$5+3i + 1 = 6+3i$$

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(___ dummy) { }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

$$5+3i \;+ 1 = 6+3i$$

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(_7_ dummy) { }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

Which type replaces blank #7 for the `dummy` parameter of post-increment `op++`?

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(_7_ dummy) { }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
       // increment real part
       // return Complex with previous value
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        // increment real part
        // return Complex with previous value
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

$$5+3i \ + 1 = 6+3i$$

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        ++real;
        // return Complex with previous value
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        ++real;
        // return Complex with previous value
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

$$5+3i + 1 = 6+3i$$

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        Complex original(___);
        ++real;
        // return Complex with previous value
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

$$5+3i\ + 1 = 6+3i$$

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        Complex original(_8_);
        ++real;
        // return Complex with previous value
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

$$5+3i \; + 1 = 6+3i$$

Which expression replaces blank #8 to instantiate a `Complex` with the same `real` and `imag` values as the current `Complex` object?

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        Complex original(_8_);
        ++real;
        // return Complex with previous value
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        // return Complex with previous value
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        // return Complex with previous value
        return ___;
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

$$5+3i \ + 1 = 6+3i$$

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        // return Complex with previous value
        return _9_;
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

Which Complex object replaces blank #9 to return the appropriate object for post-increment op++?

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        // return Complex with previous value
        return _9_;
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        // return Complex with previous value
        return original;
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

5+3i + 1 = 6+3i

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }

    // function for incrementing complex numbers (post)
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean
private:
    double real;
    double imag;
};
```

$$5+3i \; + 1 = 6+3i$$

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

now available: comp++;

equivalent to: comp.operator++(0)

A complex number class

```
class Complex {
// function for evaluating equality between complex numbers
// function for evaluating inequality between complex numbers
// function for adding complex numbers

    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

$\}$ define as non-member functions

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers

// function for evaluating inequality between complex numbers

// function for adding complex numbers
```

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```cpp
// function for evaluating equality between complex numbers
___

// function for evaluating inequality between complex numbers

// function for adding complex numbers
```

```cpp
int main() {
    Complex c1;         // 0+0i
    Complex c2(17);     // 17+0i
    Complex c3(3, -5);  // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
___

// function for evaluating inequality between complex numbers

// function for adding complex numbers
```

```
int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
___ operator==(const Complex& lhs, const Complex& rhs) { }

// function for evaluating inequality between complex numbers

// function for adding complex numbers
```

```
int main() {
    Complex c1;         // 0+0i
    Complex c2(17);     // 17+0i
    Complex c3(3, -5);  // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
_10_ operator==(const Complex& lhs, const Complex& rhs) { }

// function for evaluating inequality between complex numbers

// function for adding complex numbers
```

```
int main() {
    Complex c1;         // 0+0i
    Complex c2(17);     // 17+0i
    Complex c3(3, -5);  // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

Which return type replaces blank #10 for the `op==` function?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
_10_ operator==(const Complex& lhs, const Complex& rhs) { }

// function for evaluating inequality between complex numbers

// function for adding complex numbers
```

```
int main() {
    Complex c1;         // 0+0i
    Complex c2(17);     // 17+0i
    Complex c3(3, -5);  // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) { }

// function for evaluating inequality between complex numbers

// function for adding complex numbers
```

```
int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return _11_ && ___;
}

// function for evaluating inequality between complex numbers

// function for adding complex numbers



int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

Which condition (replacing blank #11) must be true for the `real` values of the `Complex lhs` and the `Complex rhs` for `op==` to evaluate to true?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return _11_ && ___;
}

// function for evaluating inequality between complex numbers

// function for adding complex numbers



int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && ___;
}

// function for evaluating inequality between complex numbers

// function for adding complex numbers



int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```cpp
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers

// function for adding complex numbers



int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```cpp
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}
```

compilation error

```cpp
// function for evaluating inequality between complex numbers

// function for adding complex numbers
```

```cpp
int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

How can the `op==` non-member function be given access to private member variables of `Complex` objects?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}
```

compilation error

```
// function for evaluating inequality between complex numbers

// function for adding complex numbers



int main() {
    Complex c1;         // 0+0i
    Complex c2(17);     // 17+0i
    Complex c3(3, -5);  // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    // make operator== friend of class

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}
```

compilation error

```
// function for evaluating inequality between complex numbers

// function for adding complex numbers
```

```
int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    // make operator== friend of class

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```cpp
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}
```

compilation error

```cpp
// function for evaluating inequality between complex numbers

// function for adding complex numbers



int main() {
    Complex c1;         // 0+0i
    Complex c2(17);     // 17+0i
    Complex c3(3, -5);  // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```cpp
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}
```

~~compilation error~~

```cpp
// function for evaluating inequality between complex numbers

// function for adding complex numbers



int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```cpp
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers

// function for adding complex numbers



int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 == c2 ? "==" : "!=") << " c2\n";
    cout << "c1 " << (c1 == c1 ? "==" : "!=") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
---

// function for adding complex numbers


int main() {
    Complex c1;         // 0+0i
    Complex c2(17);     // 17+0i
    Complex c3(3, -5);  // 3-5i

    cout << "c1 " << (c1 != c2 ? "!=" : "==") << " c2\n";
    cout << "c1 " << (c1 != c1 ? "!=" : "==") << " c1\n";
}
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
---

// function for adding complex numbers


int main() {
    Complex c1;         // 0+0i
    Complex c2(17);     // 17+0i
    Complex c3(3, -5);  // 3-5i

    cout << "c1 " << (c1 != c2 ? "!=" : "==") << " c2\n";
    cout << "c1 " << (c1 != c1 ? "!=" : "==") << " c1\n";
}
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
---

// function for adding complex numbers


int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 != c2 ? "!=" : "==") << " c2\n";
    cout << "c1 " << (c1 != c1 ? "!=" : "==") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) { }

// function for adding complex numbers
```

```
int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 != c2 ? "!=" : "==") << " c2\n";
    cout << "c1 " << (c1 != c1 ? "!=" : "==") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```cpp
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return ___;
}

// function for adding complex numbers

int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 != c2 ? "!=" : "==") << " c2\n";
    cout << "c1 " << (c1 != c1 ? "!=" : "==") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return _12_;
}

// function for adding complex numbers

int main() {
    Complex c1;         // 0+0i
    Complex c2(17);     // 17+0i
    Complex c3(3, -5);  // 3-5i

    cout << "c1 " << (c1 != c2 ? "!=" : "==") << " c2\n";
    cout << "c1 " << (c1 != c1 ? "!=" : "==") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

Which expression (replacing blank #12) utilizing `==` will evaluate to `true` when the `Complex lhs` and `Complex rhs` are not equal?

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```cpp
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return _12_;
}

// function for adding complex numbers

int main() {
    Complex c1;         // 0+0i
    Complex c2(17);     // 17+0i
    Complex c3(3, -5);  // 3-5i

    cout << "c1 " << (c1 != c2 ? "!=" : "==") << " c2\n";
    cout << "c1 " << (c1 != c1 ? "!=" : "==") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers

int main() {
    Complex c1;          // 0+0i
    Complex c2(17);      // 17+0i
    Complex c3(3, -5);   // 3-5i

    cout << "c1 " << (c1 != c2 ? "!=" : "==") << " c2\n";
    cout << "c1 " << (c1 != c1 ? "!=" : "==") << " c1\n";
}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
c1 != c2
c1 == c1
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

    _13_ operator+=(const Complex& rhs) { }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

Which return type replaces blank #13 when implementing op+=?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }

    _13_ operator+=(const Complex& rhs) { }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) { }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        ___ // update real using rhs real value
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

Which statement will update the current `Complex` object's `real` value to the **sum** of its `real` value and `Complex rhs`'s `real` value?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        ___ // update real using rhs real value
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        ---
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        ___
    }

...
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return ___;
    }

...
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return _14_;
    }

...
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

Which expression (replacing blank #14) will result in the current object being returned by op+=?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return _14_;
    }

...
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }

    friend bool operator==(const Complex& lhs, const Complex& rhs);

public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

...
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```cpp
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
```

Let's implement op+= first

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
___
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
___ operator+(const Complex& lhs, const Complex& rhs) { }
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
_15_ operator+(const Complex& lhs, const Complex& rhs) { }
```

5+3i + 2-1i = 7+2i

Which return type replaces blank #15 so that a new `Complex` object (not `lhs` or `rhs`) is returned by `op+`?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
_15_ operator+(const Complex& lhs, const Complex& rhs) { }
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) { }
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    // make a copy of lhs
    // add rhs to copy
    // return the copy
}
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    // make a copy of lhs
    Complex ___;
    // add rhs to copy
    // return the copy
}
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    // make a copy of lhs
    Complex _16_;
    // add rhs to copy
    // return the copy
}
```

5+3i + 2-1i = 7+2i

Which expression replaces blank #16 to create a copy of `Complex lhs` named `result`?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    // make a copy of lhs
    Complex _16_;
    // add rhs to copy
    // return the copy
}
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    // make a copy of lhs
    Complex result(lhs);
    // add rhs to copy
    // return the copy
}
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    // add rhs to copy
    ___
    // return the copy
}
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    // add rhs to copy
    _17_
    // return the copy
}
```

5+3i + 2-1i = 7+2i

Which statement will update `result` to the **sum** of `result` and `rhs`?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    // add rhs to copy
    _17_
    // return the copy
}
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

utilizing op+=
member function

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    // return the copy
}
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    // return the copy
}
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean

private:
    double real;
    double imag;
};
```

```
// function for evaluating equality between complex numbers
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

// function for evaluating inequality between complex numbers
bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

// function for adding complex numbers
Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}
```

5+3i + 2-1i = 7+2i

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

// function for converting complex number to boolean
---

private:
    double real;
    double imag;
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    if (zero)
        cout << "zero is true\n";
    else
        cout << "zero is false\n";

}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
zero is false
```

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    operator bool() ___ { }

private:
    double real;
    double imag;
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```cpp
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    if (zero)
        cout << "zero is true\n";
    else
        cout << "zero is false\n";

}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
zero is false
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    operator bool() const { }

private:
    double real;
    double imag;
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    if (zero)
        cout << "zero is true\n";
    else
        cout << "zero is false\n";

}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
zero is false
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    operator bool() const { }

private:
    double real;
    double imag;
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    if (zero)
        cout << "zero is true\n";
    else
        cout << "zero is false\n";

}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
zero is false
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    operator bool() const {
        return ___ || ___;
    }

...
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    if (zero)
        cout << "zero is true\n";
    else
        cout << "zero is false\n";

}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
zero is false
```

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    operator bool() const {
        return _17_ || ___;
    }

...
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```cpp
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    if (zero)
        cout << "zero is true\n";
    else
        cout << "zero is false\n";

}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
zero is false
```

Which condition replaces blank #17 such that it evaluates to true when the `real` part of the `Complex` is not equal to `0`?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    operator bool() const {
        return _17_ || ___;
    }

...
};
```

Complex as bool type
- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    if (zero)
        cout << "zero is true\n";
    else
        cout << "zero is false\n";

}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
zero is false
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    operator bool() const {
        return (real != 0) || ___;
    }

...
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    if (zero)
        cout << "zero is true\n";
    else
        cout << "zero is false\n";

}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
zero is false
```

A complex number class

```cpp
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    operator bool() const {
        return (real != 0) || ___;
    }

...
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```cpp
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    if (zero)
        cout << "zero is true\n";
    else
        cout << "zero is false\n";

}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
zero is false
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    operator bool() const {
        return (real != 0) || (imag != 0);
    }
...
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    if (zero)
        cout << "zero is true\n";
    else
        cout << "zero is false\n";

}
```

```
% g++ -std=c++11 aoo.cpp -o aoo.o
% ./aoo.o
zero is false
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    operator bool() const {
        return (real != 0) || (imag != 0);
    }

...
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex comp(0, 2);

    ~~int num = comp + 10;~~

}
```

implicit conversion

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    ___ operator bool() const {
        return (real != 0) || (imag != 0);
    }

...
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    int num = zero + 10;

}
```

~~int num = zero + 10;~~ *implicit conversion*

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    _18_ operator bool() const {
        return (real != 0) || (imag != 0);
    }

...
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    int num = zero + 10;

}
```

~~int num = zero + 10;~~ *implicit conversion*

Which keyword (replacing blank #18) when added to a function definition prevents implicit conversions?

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    _18_ operator bool() const {
        return (real != 0) || (imag != 0);
    }

...
};
```

Complex as bool type
- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    ~~int num = zero + 10;~~
}
```

implicit conversion

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    // function for converting complex number to boolean
    explicit operator bool() const {
        return (real != 0) || (imag != 0);
    }

...
};
```

Complex as bool type

- false when value is 0+0i
- true otherwise

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}

int main() {
    Complex zero(0);

    ~~int num = zero + 10;~~

}
```

implicit conversion

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    explicit operator bool() const {
        return (real != 0) || (imag != 0);
    }

...
};
```

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator++() {
        ++real;
        return *this;
    }
    Complex operator++(int dummy) {
        Complex original(*this);
        ++real;
        return original;
    }
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    explicit operator bool() const {
        return (real != 0) || (imag != 0);
    }
private:
    double real;
    double imag;
};
```

```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}

bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}
```

A complex number class

```
class Complex {
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        return os;
    }
    friend bool operator==(const Complex& lhs, const Complex& rhs);
public:
    Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}
    Complex& operator+=(const Complex& rhs) {
       real += rhs.real;
       imag += rhs.imag;
       return *this;
    }
    Complex& operator++() {
       ++real;
       return *this;
    }
    Complex operator++(int dummy) {
       Complex original(*this);
       ++real;
       return original;
    }
    explicit operator bool() const {
       return (real != 0) || (imag != 0);
    }
private:
    double real;
    double imag;
};
```

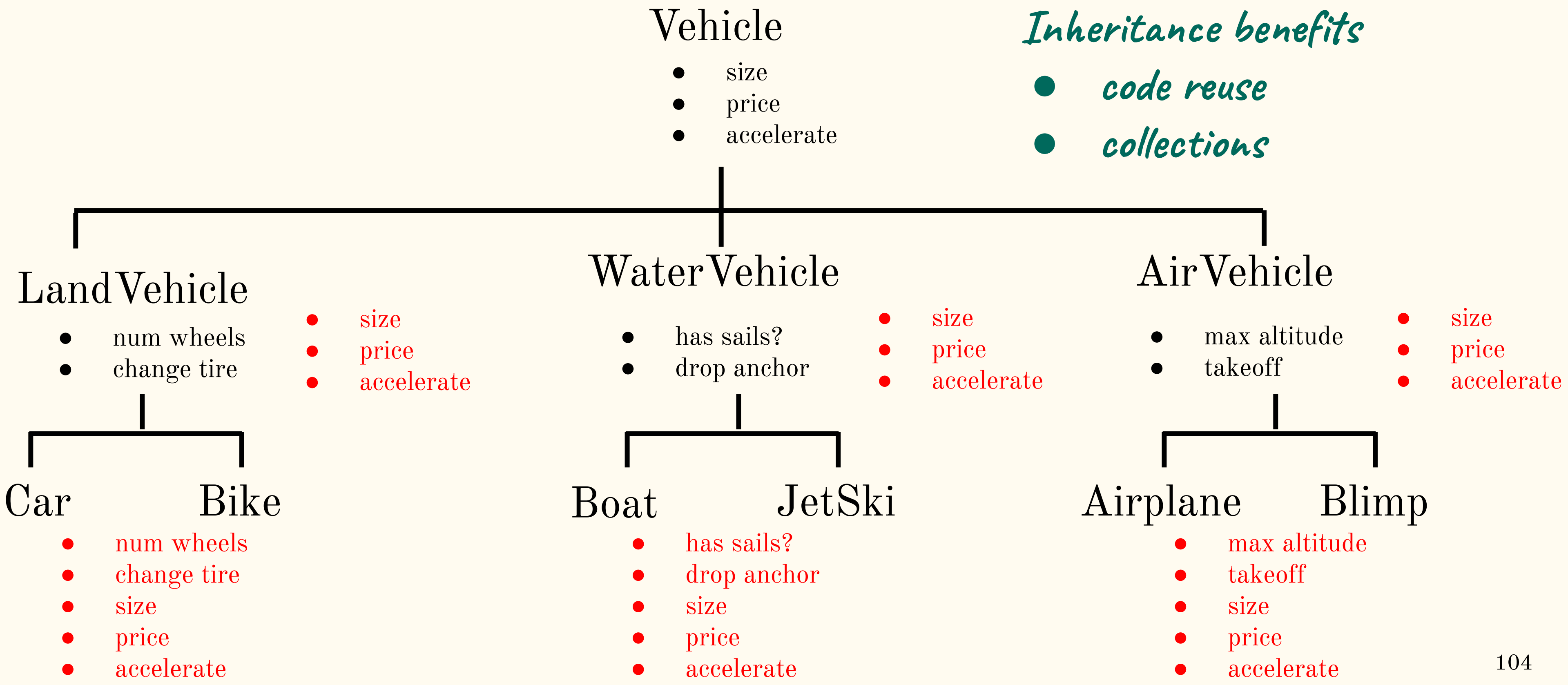
```
bool operator==(const Complex& lhs, const Complex& rhs) {
    return lhs.real == rhs.real && lhs.imag == rhs.imag;
}


bool operator!=(const Complex& lhs, const Complex& rhs) {
    return !(lhs == rhs);
}

Complex operator+(const Complex& lhs, const Complex& rhs) {
    Complex result(lhs);
    result += rhs;
    return result;
}
```

Background on Inheritance

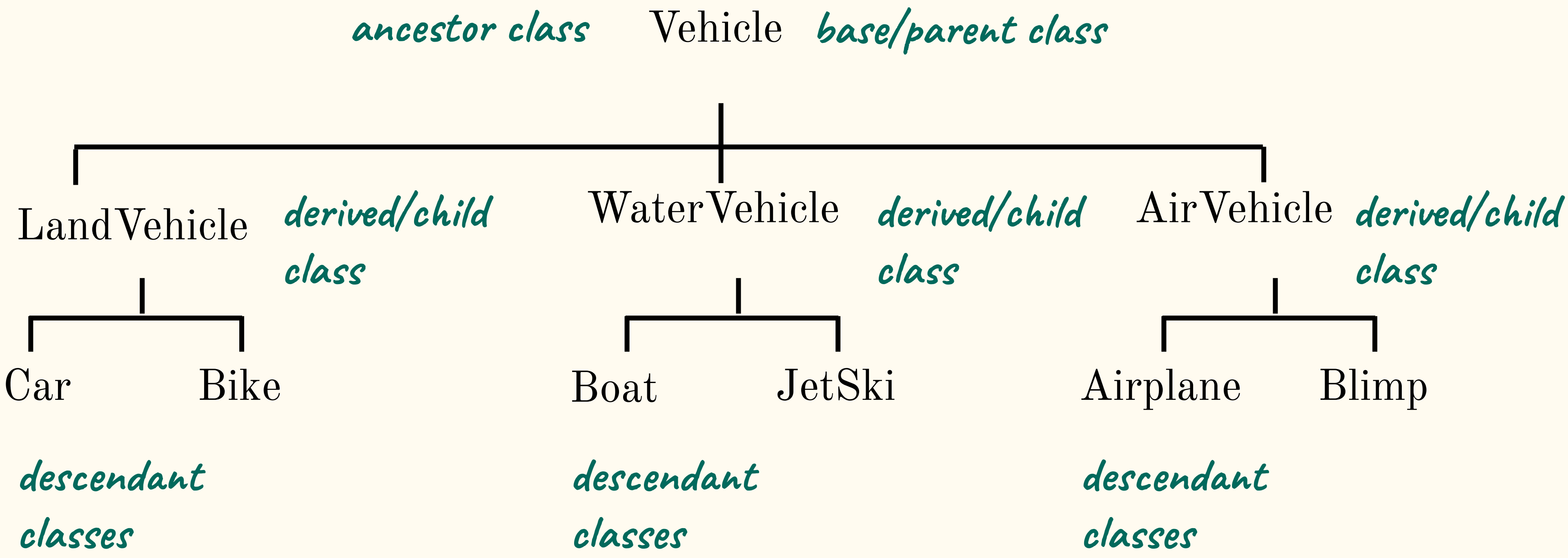
Inheritance



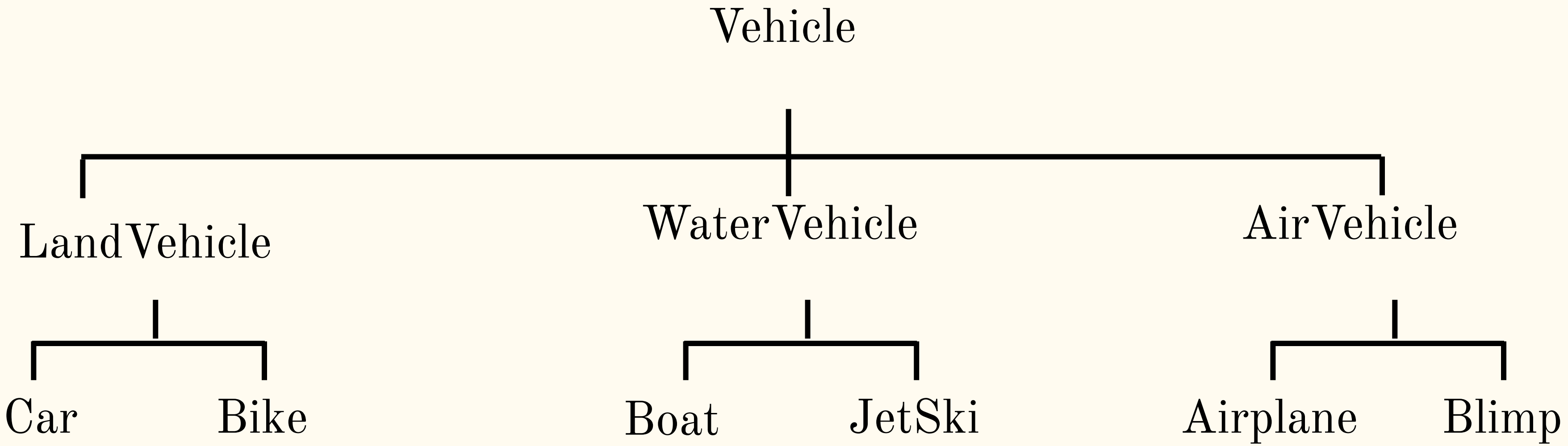
Inheritance benefits

- *code reuse*
- *collections*

Inheritance terminology

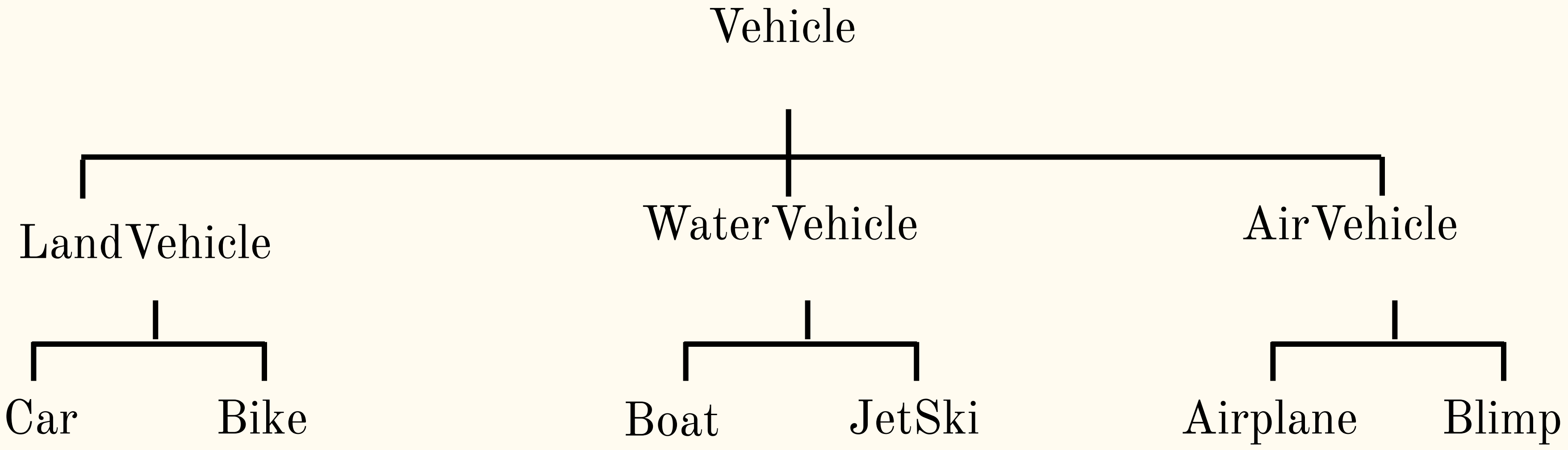


Inheritance relationships



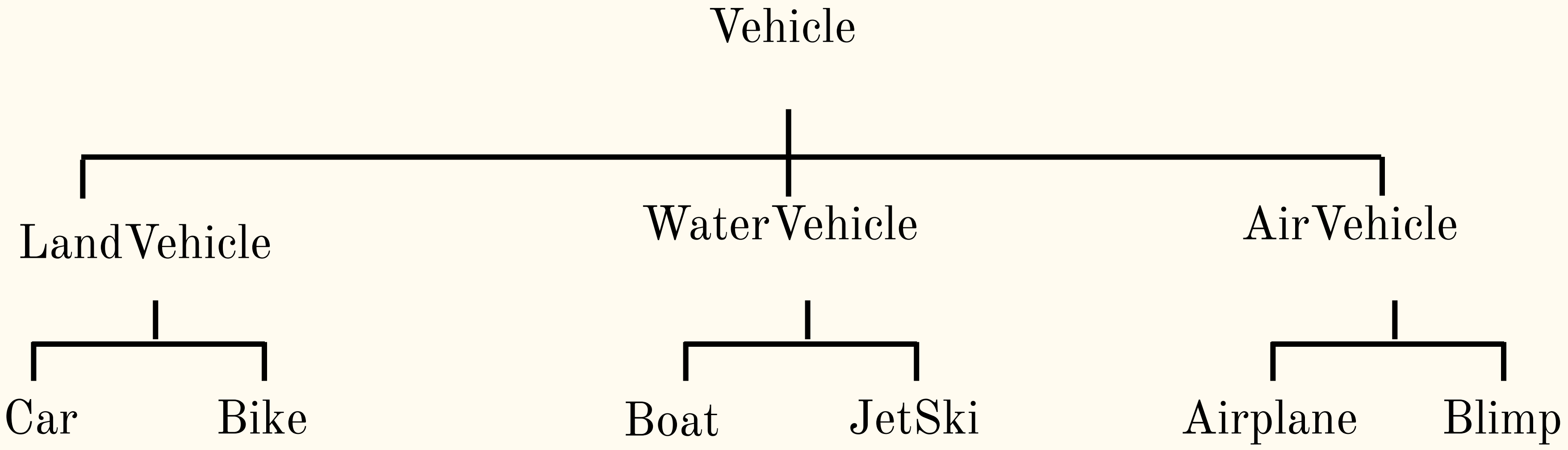
________ *is a* ________.

Inheritance relationships



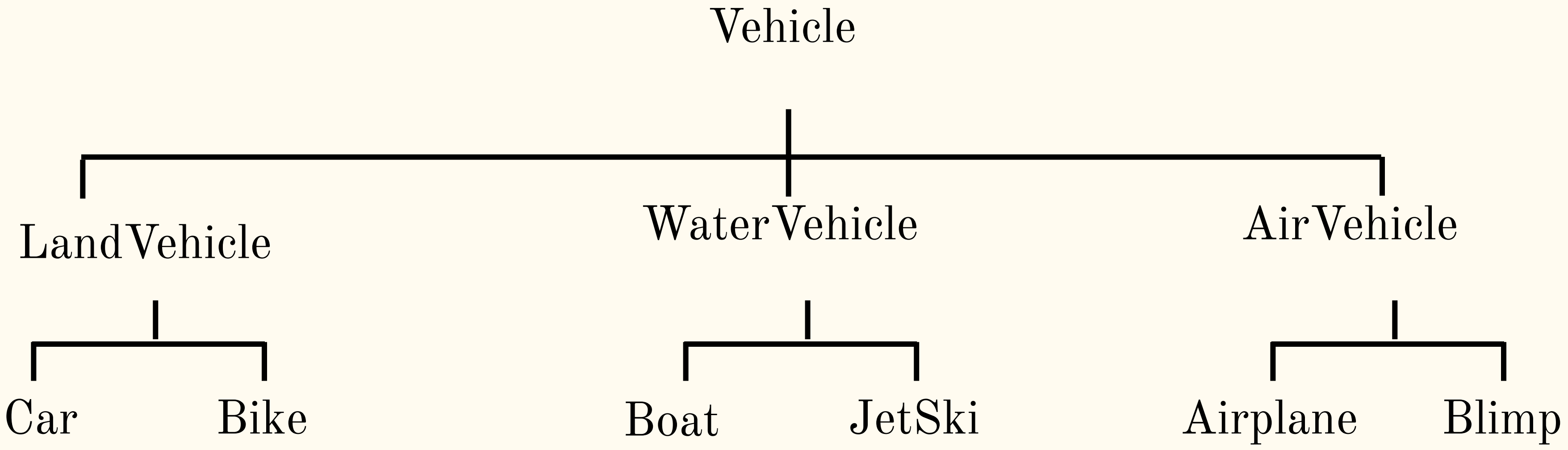
LandVehicle is a Vehicle.

Inheritance relationships



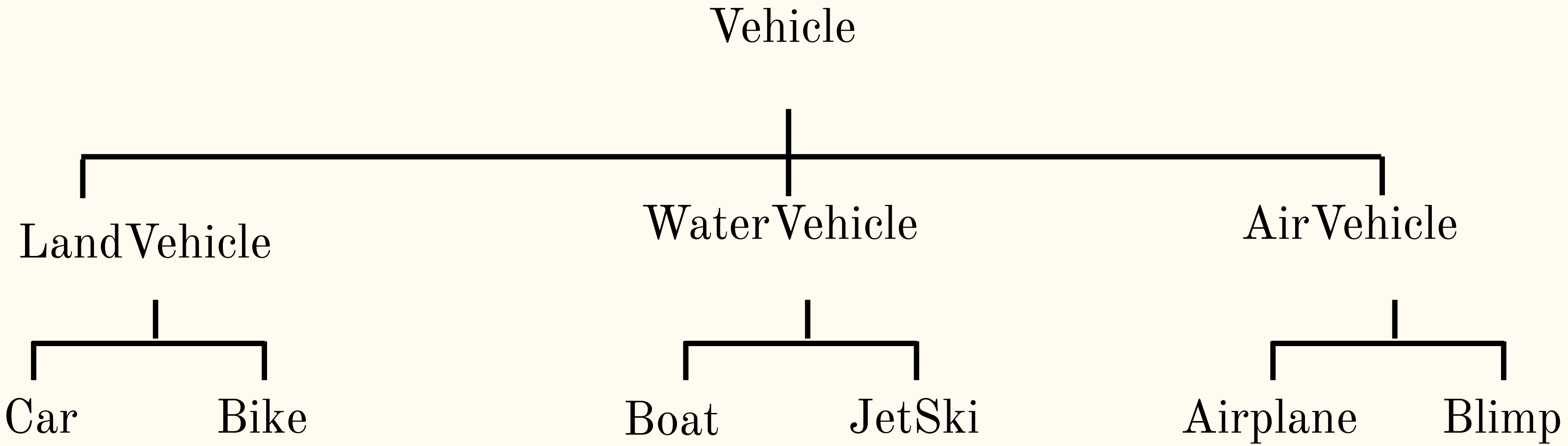
Car is a Vehicle.

Inheritance relationships



Blimp ~~is a~~ WaterVehicle .

Inheritance in code



```
class Vehicle { };
class LandVehicle : public Vehicle { };
class WaterVehicle : public Vehicle { };
class JetSki : public WaterVehicle { };
```

public keyword required

Inheritance basics

Inheriting methods

```
class Animal {};

class Lion : public Animal {};

class Tiger : public Animal {};

class Bear : public Animal {};
```

Inheriting methods

inherited from base class

```
class Animal {
public:
    void eat() { cout << "Animal eating\n"; }
};

class Lion : public Animal {};

class Tiger : public Animal {};

class Bear : public Animal {};

int main() {
    Bear yogi;
    yogi.eat();
}
```

no eat() method defined

```
Animal eating
```

Overriding methods

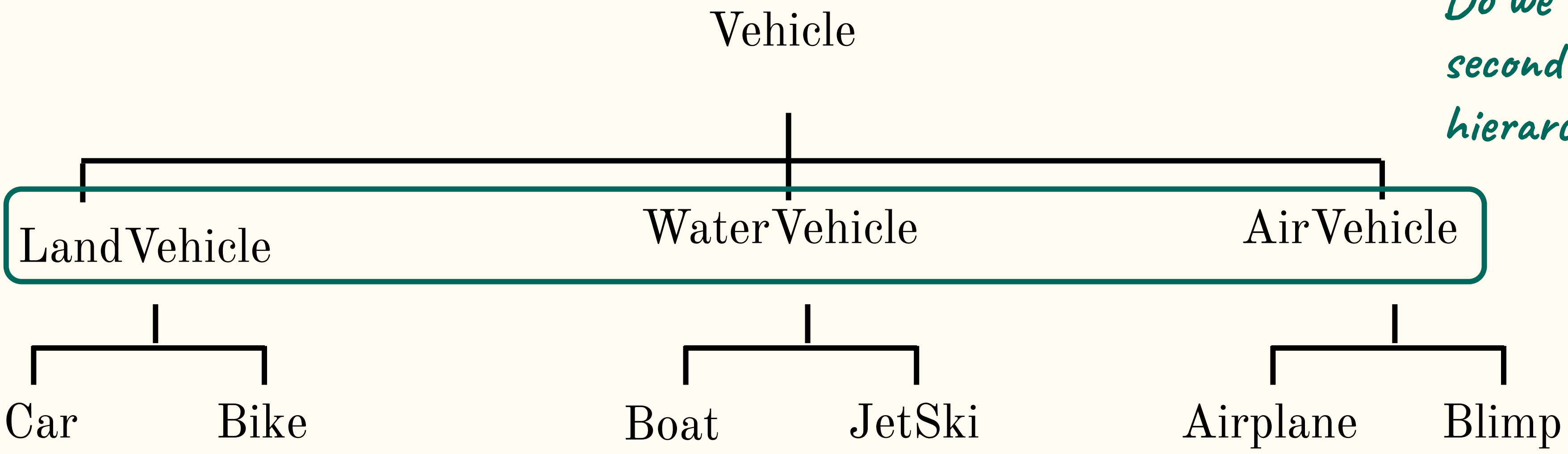
```
class Animal {
public:
    void eat() { cout << "Animal eating\n"; }
};
class Lion : public Animal {};
class Tiger : public Animal {
public:
    void eat() { cout << "Tiger eating\n"; }
};
class Bear : public Animal {};

int main() {
    Bear yogi;
    yogi.eat();

    Tiger tigger;
    tigger.eat();
}
```

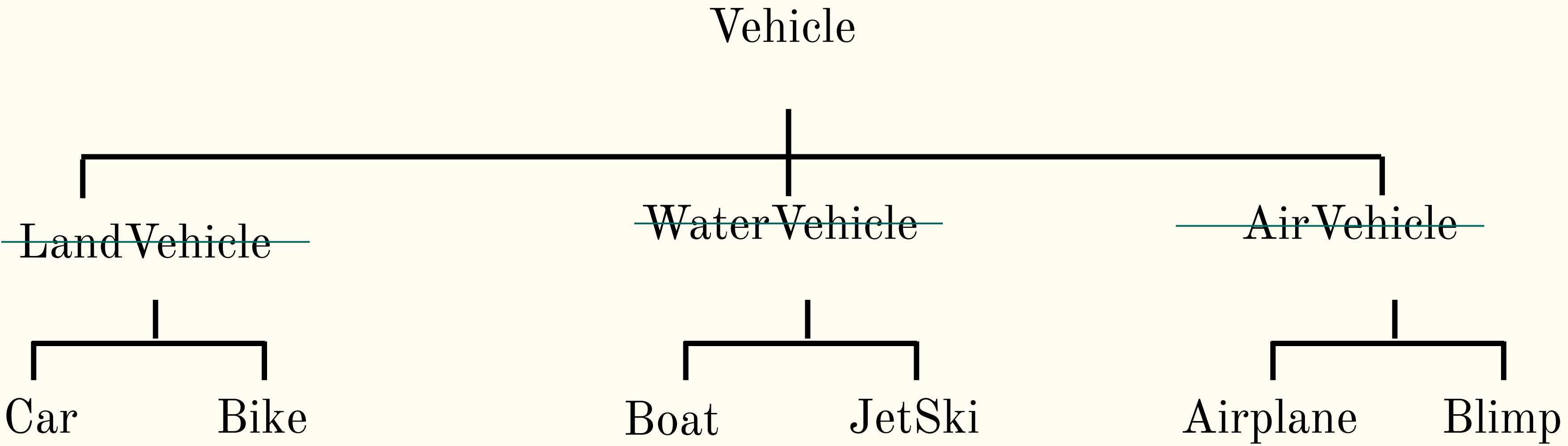
```
Animal eating
Tiger eating
```

Principle of Substitutability

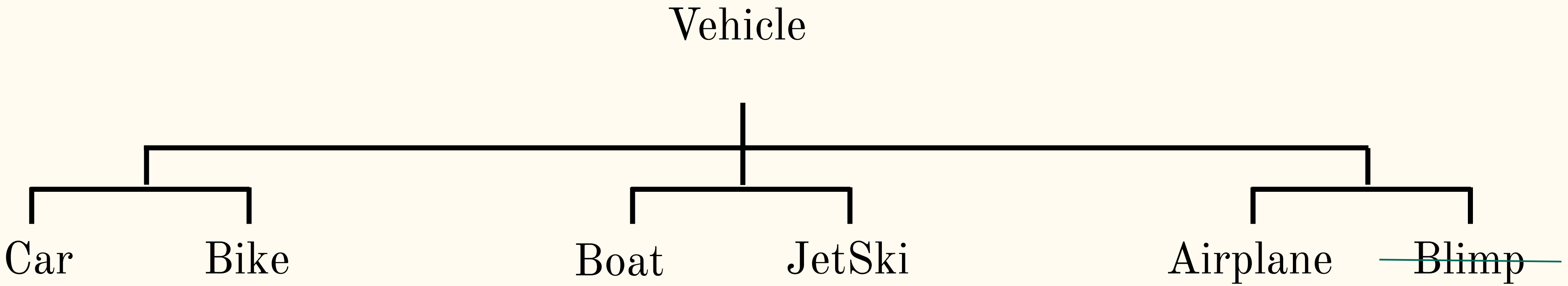


Do we need second level of hierarchy?

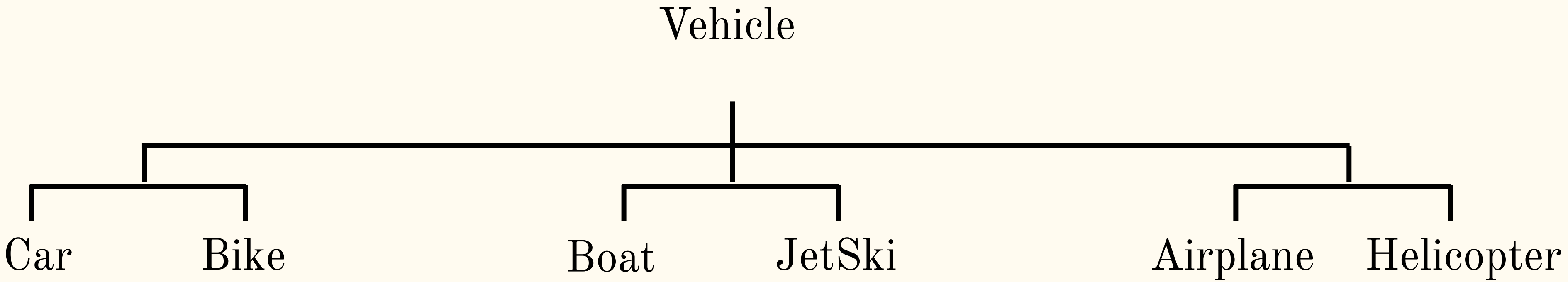
Principle of Substitutability



Principle of Substitutability

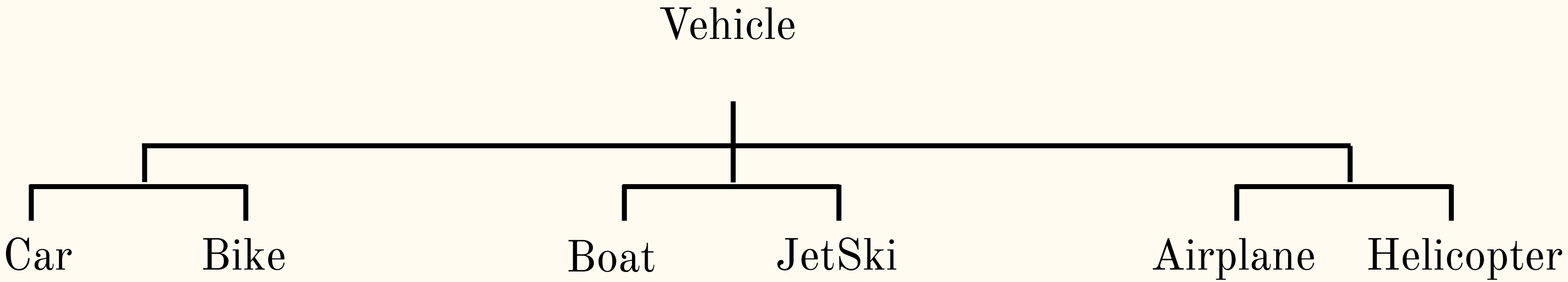


Principle of Substitutability



```
class Vehicle { };
class Car : public Vehicle { };
class Helicopter : public Vehicle { };
```

Principle of Substitutability



```
class Vehicle {
public:
    void roll() { cout << "rolling"; }
};
class Car : public Vehicle { };
class Helicopter : public Vehicle { };
```

```
void move(Vehicle& v) {
    v.roll();
}
```

Any Vehicle or derived type should be a valid argument

Can a car roll? Sure!

Can a helicopter roll? Not without some help...

Principle of Substitutability

Inheritance design violates Principle of Substitutability

Potential solutions:

1) *Remove roll method from Vehicle class*
2) *Define Helicopter such that it does not inherit from Vehicle class*

```
class Vehicle {
public:
    void roll() { cout << "rolling"; }
};
class Car : public Vehicle { };
class Helicopter : public Vehicle { };
```

```
void move(Vehicle& v) {
    v.roll();
}
```