**SRS Setup**
**Login: student.turningtechnologies.com**
**Session ID: 20220321<A|D>**

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

# Operator Overloading

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

# Agenda

- Operator overloading review
- In-class problem

## Operator overloading review

# Operator overloading limitations

- changing the meaning of operators for built-in types
  - `1 + 1 == 2` *always true*
- changing precedence
  - `2 + 3 * 4` ** always evaluated before +*
- creating new operators
  - `2 ** 3` *** operator non-existent in C++*
- overloading ternary "conditional" operator (?)
  - test() ? a : b *not possible*
- changing operator "associativity"
  - `a + b + c` *evaluated left-to-right*
  - `a = b = c` *evaluated right-to-left*
- changing arity *i.e., number of operands involved*
  - binary: <<, %, ==, etc
  - unary: !, &, ++, etc
- changing order of evaluation/short circuiting behavior
  - `f() && g()` *g not evaluated when f evaluates to false*
  - `f() || g()` *g not evaluated when f evaluates to true*

# operator<<

```
class Cat {

    friend ostream& operator<<(ostream&, const Cat&);
public:
    Cat(const string& the_name, const string& the_color, double the_weight)
    : name(the_name), weight(the_weight), color(the_color) {}

private:
    string name;
    string color;
    double weight;
};
```

```
ostream& operator<< (ostream& os, const Cat& rhs) {
    os << "Displaying a Cat named" << rhs.name << " with color ";
    os << rhs.color << " and weight " << rhs.weight << endl;
    return os;
}
```

```
int main() {
    Cat my_cat(
        "Whiskers",
        "brown",
        8
    );

    cout << my_cat << endl;
}
```

```
Displaying a Cat named Whiskers with
color brown and weight 8
```

## operator=

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```

Requirements of assignment operator

- check for self-assignment
- free old memory (if needed)
- allocate new memory (if needed)
- copy values
- return proper type and object

```
class Vector {
public:
    ...
    Vector& operator=(const Vector& rhs) {
        if (this != &rhs) {
            delete [] data;
            data = new int[the_capacity];
            the_size = rhs.the_size;
            the_capacity = rhs.the_capacity;
            for (size_t i = 0; i < the_size; ++i) {
                data[i] = rhs.data[i];
            }
        }
        return *this;
    }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

# operator[]

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

    vec[1] = -5;

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

    int& operator[](size_t i) { return data[i]; }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

*MUST be implemented as a member function*

# Operator expressions to functions

```
Elephant el1, el2;

el1 + el2
```

convert to
function call

```
class Elephant {

    ...

    Elephant& operator+(const Elephant&)

    ...
};
```

*member*

*non-member*

```
Elephant operator+(const Elephant&, const Elephant&)
```

## Comparison operators

`==  !=  <  >  <=  >=`

| **property** | **value** |
|---|---|
| arity | binary |
| member/non-member | either |
| return type | bool |

## Arithmetic operators

`+  -  *  /  %`

| **property** | **value** | **notes** |
|---|---|---|
| arity | operator specific | binary only (/ % *)<br>unary and binary (+ -) |
| member/non-member | either | |
| return type | by value | |

# pre-increment `operator++`

- increment and return *modified* value

```
int num = 0;

int num2 = ++num;
```

num now 1

num2 assigned 1

```
Elephant& Elephant::operator++() {
    ++weight;
    return *this;
} // member implementation example
```

| **property** | **value** | **notes** |
|---|---|---|
| arity | unary | |
| member/non-member | either | |
| return type | by reference | |

pre-decrement operator-- works similarly

## post-increment `operator++`

- increment and return value (prior to increment)

```
int num = 0;

int num2 = num++;
```

num now 1

num2 assigned 0

| property | value |
|---|---|
| arity | unary |
| member/non-member | either |
| return type | by value |

```
Elephant Elephant::operator++(int dummy) {
      Elephant original(*this);

      ++weight;
      return original;
} //member implementation example
```

`int dummy`: used to distinguish from pre-increment function

post-decrement operator-- works similarly

## assignment operators (other than `=`)

`+=  -=  *=  /=  %=`

(Recommended) implementation

- member function
- returns reference to left hand object

| **property** | **value** |
|---|---|
| arity | binary |
| member/non-member | member |
| return type | by reference |

```
Elephant& Elephant::operator-=(const Elephant& rhs) {
        weight -= rhs.weight;

        return *this;
}
```

## Implementing operators from others

- operators can call other overloaded operators

```
bool operator==(const Elephant& lhs, const Elephant& rhs) {
      // implementation of == operator
}

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

## Implementing operators from others

- operators can call other overloaded operators

```
Elephant& Elephant::operator+=(const Elephant& rhs) {
        weight += rhs.weight;

        return *this;
}
Elephant operator+ (const Elephant& lhs, const Elephant& rhs) {
      Elephant temp = lhs;
      return temp += rhs;
}
```

## Boolean type conversion

```
int main() {
    ifstream jab("jabberwocky");

    if (!jab) {   possible to implement with operator!()
        cerr << "failed to open jabberwocky";
        exit(1);
    }
    ...               consider...
}
```

# Boolean type conversion

```
int main() {
    ifstream jab("jabberwocky");

    if (!jab) {
        cerr << "failed to open jabberwocky";
        exit(1);
    }

    string something;
    while (jab >> something) {
        cout << something << endl;
    }
    jab.close();
}
```

*possible to implement with operator!()*

- `jab >> something` also evaluates to boolean value
- `operator>>()` returns `ifstream&`
- `ifstream` evaluated as boolean value

*How??*

## Boolean type conversion

```
class ifstream {
        …
        public:
            ...
            operator bool() const;
        ...
};
```

ifstream not modified

no return type

returns true when ifstream is ok to read from; false otherwise

space in definition

# Implicit vs. explicit conversion

```
class Elephant {
    ...
    public:
        operator bool() const { return true; }
        ...
        // operator+(int) not defined
    ...
};
```

```
int main() {
    Elephant el1;
    cout << el1 + 1 << endl;
}
```

~~compilation error?~~

$\underbrace{\texttt{el1 + 1}}_{2}$

a boolean value can be *implicitly* converted into an integer

$true \longrightarrow 1$

$false \longrightarrow 0$

## Implicit vs. explicit conversion

```
class Elephant {
    ...
    public:
        operator bool() const { return true; }
        ...
        // operator+(int) not defined
    ...
};
```

```
int main() {
    Elephant el1;
    cout << el1 + 1 << endl;
}
```

~~compilation error?~~

# Implicit vs. explicit conversion

```
class Elephant {
    ...
    public:
        explicit operator bool() const { return true; }
        ...
        // operator+(int) not defined
    ...
};

int main() {
    Elephant el1;
    cout << el1 + 1 << endl;
}
```

*compilation error?* ✔

adding `explicit` keyword prevents implicit conversion

- operator called when boolean expected

## Implicit vs. explicit conversion

```
class Elephant {
    ...
    public:
        explicit operator bool() const { return true; }
        ...
        // operator+(int) not defined
    ...
};
```

adding `explicit` keyword prevents implicit conversion
-- operator called when boolean expected

```
int main() {
    Elephant el1;
    if (el1) { ... }   ✔
}
```

## Member vs. non-member function

- input (`>>`) and output (`<<`) operators typically implemented as non-member functions
- assignment/combination operators (`+=`, `-=`, etc) typically implemented as members
- other binary operators (`%`, `/`, `==`, etc) typically implemented as non-members
- unary operators (`!`, `++`, `--`, etc) typically implemented as member functions

## `friend` status for non-member operator functions

- Some advantages and disadvantages
  - advantage: friend modifier reduces need for accessor and mutator methods (helps code readability)
  - disadvantage: update to class may require modification of `friend` function
- Operators overloaded as non-members typically defined as `friend` functions
  - input (`operator>>`) and output (`operator<<`)
  - less than (`operator<`) and equality (`operator==`)
    - remaining operators (`!=`, `<=`, `>` and `>=`) built from `<` and `==` without friend status

## In-class problem

## Complex numbers

real numbers

imaginary number

$$\underbrace{a}_{\text{real part}} + \underbrace{bi}_{\text{imaginary part}}$$

$i^2 = -1$

# A complex number class

Properties

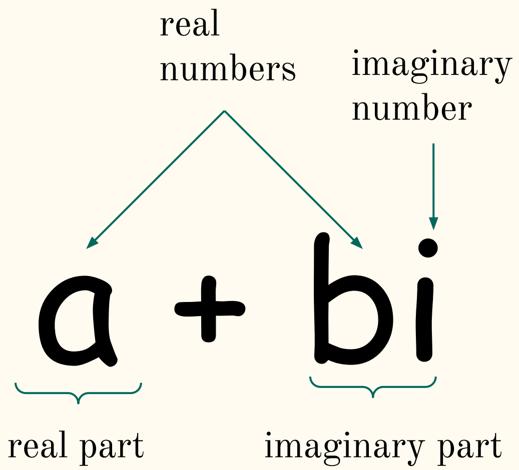
- real part
- imaginary part

*define corresponding member variables*

Behavior

- instantiation
- output
- addition
- comparison
- type conversion
- etc

*define corresponding functions (member and non-member)*



$i^2 = -1$

# A complex number class

```
class Complex {
// variables for properties: real part, imaginary part
};
```

Properties

- real part ✓
- imaginary part ✓

Behavior

- instantiation
- output
- addition
- comparison
- type conversion
- etc

## A complex number class

Properties

- real part ✔
- imaginary part ✔

Behavior

- instantiation ✔
- output
- addition
- comparison
- type conversion
- etc

```
class Complex {
// constructor(s) for instantiation

// variables for properties: real part, imaginary part
};
```

# A complex number class

Properties
- real part ✓
- imaginary part ✓

Behavior
- instantiation ✓
- output ✓
- addition
- comparison
- type conversion
- etc

```
class Complex {
// function for inserting into output stream

// constructor(s) for instantiation

// variables for properties: real part, imaginary part
};
```

# A complex number class

Properties

- real part ✔
- imaginary part ✔

Behavior

- instantiation ✔
- output ✔
- addition
- comparison
- type conversion
- etc

```
class Complex {
// function for adding complex numbers

// function for inserting into output stream

// constructor(s) for instantiation

// variables for properties: real part, imaginary part
};
```

# A complex number class

Properties

- real part ✔
- imaginary part ✔

Behavior

- instantiation ✔
- output ✔
- addition
- comparison
- type conversion
- etc

```
class Complex {
// function for adding complex numbers

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// variables for properties: real part, imaginary part
};
```

# A complex number class

Properties

- real part ✓
- imaginary part ✓

Behavior

- instantiation ✓
- output ✓
- addition ✓
- comparison
- type conversion
- etc

```
class Complex {
// function for adding complex numbers

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// variables for properties: real part, imaginary part
};
```

# A complex number class

Properties
- real part ✓
- imaginary part ✓

Behavior
- instantiation ✓
- output ✓
- addition ✓
- comparison
- type conversion
- etc

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

// function for adding complex numbers

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// variables for properties: real part, imaginary part
};
```

# A complex number class

Properties
- real part ✔
- imaginary part ✔

Behavior
- instantiation ✔
- output ✔
- addition ✔
- comparison ✔
- type conversion
- etc

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

// function for evaluating inequality between complex numbers

// function for adding complex numbers

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// variables for properties: real part, imaginary part
};
```

# A complex number class

Properties
- real part ✓
- imaginary part ✓

Behavior
- instantiation ✓
- output ✓
- addition ✓
- comparison ✓
- type conversion ✓
- etc

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

// function for evaluating inequality between complex numbers

// function for adding complex numbers

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// function for converting complex number to boolean

// variables for properties: real part, imaginary part
};
```

## A complex number class

Properties

- real part ✓
- imaginary part ✓

Behavior

- instantiation ✓
- output ✓
- addition ✓
- comparison ✓
- type conversion ✓
- etc

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

// function for evaluating inequality between complex numbers

// function for adding complex numbers

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// function for converting complex number to boolean

// variables for properties: real part, imaginary part
};
```

## 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

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// function for converting complex number to boolean

// variables for properties: real part, imaginary part
};
```

## 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

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// function for converting complex number to boolean

// variables for properties: real part, imaginary part
private:
    ___ real;
    ___ imag;
};
```

## 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

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// function for converting complex number to boolean

// variables for properties: real part, imaginary part
private:
    _1_ real;
    _1_ imag;
};
```

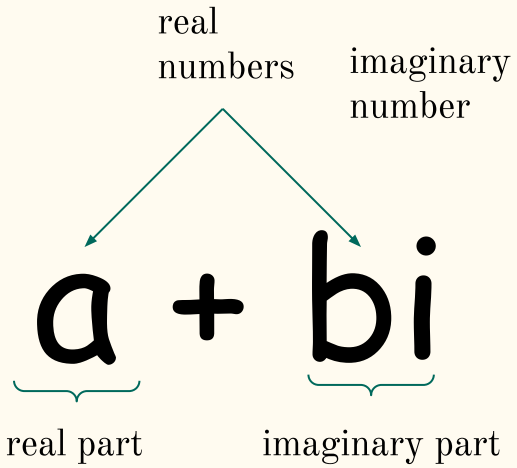
# TurningPoint

**SRS Setup**
**Login: student.turningtechnologies.com**
**Session ID: 20220321<A|D>**

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

## Which type replaces blank #1 for declaring the private member variables of the `Complex` class?

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

// function for evaluating inequality between complex numbers

// function for adding complex numbers

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// function for converting complex number to boolean

// variables for properties: real part, imaginary part
private:
    _1_ real;
    _1_ imag;
};
```



## 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

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// function for converting complex number to boolean

// variables for properties: real part, imaginary part
private:
    double real;
    double imag;
};
```

# 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

// function for inserting into output stream

// constructor(s) for instantiation

// function for incrementing complex numbers

// function for converting complex number to boolean

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

## 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

// function for inserting into output stream

// constructor(s) for instantiation
---

// function for incrementing complex numbers

// function for converting complex number to boolean

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

*How to initialize values?*

# 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

// function for inserting into output stream

// constructor(s) for instantiation
Complex() : real(0), imag(0) {}

// function for incrementing complex numbers

// function for converting complex number to boolean

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

*How to initialize values?*

- modifying `real` and/or `imag` requires mutator or operator overloading
- initializing to desired value should be supported

## 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

// function for inserting into output stream

// constructor(s) for instantiation
Complex(double real, double imag) : real(real), imag(imag) {}

// function for incrementing complex numbers

// function for converting complex number to boolean

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

*How to initialize values?*

- forcing arguments to be provided
- `Complex comp = Complex();` not supported

## 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

// function for inserting into output stream

// constructor(s) for instantiation
Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}

// function for incrementing complex numbers

// function for converting complex number to boolean

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

*How to initialize values?*

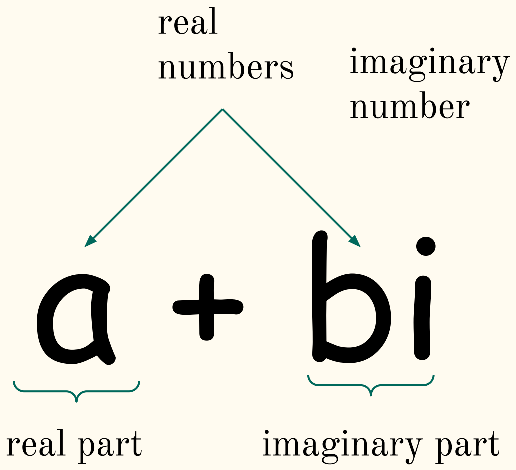
- single constructor supports
  - `Complex comp1 = Complex();`
  - `Complex comp2 = Complex(5);` // a real number
  - `Complex comp3 = Complex(5, 2);` // real and imaginary parts

## 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

// function for inserting into output stream

// constructor(s) for instantiation
Complex(double real = 0, double imag = 0) : real(real), imag(imag) {}

// function for incrementing complex numbers

// function for converting complex number to boolean

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

# 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

// function for inserting into output stream
---

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

// function for incrementing complex numbers

// function for converting complex number to boolean

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



any real number can be expressed as complex number

## How can the real number 5 be expressed as a complex number?

real numbers

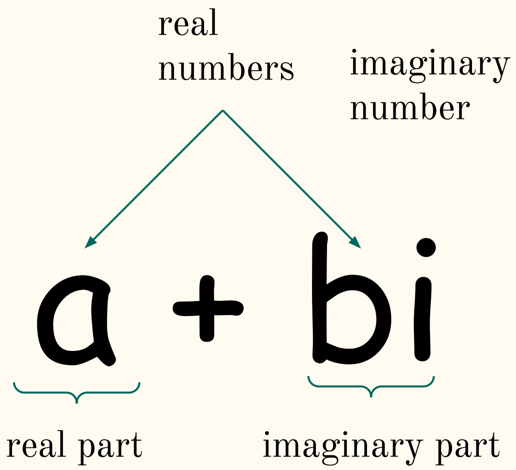
imaginary number

$$\underbrace{a}_{\text{real part}} + \underbrace{bi}_{\text{imaginary part}}$$

*any real number can be expressed as complex number*

## A complex number class

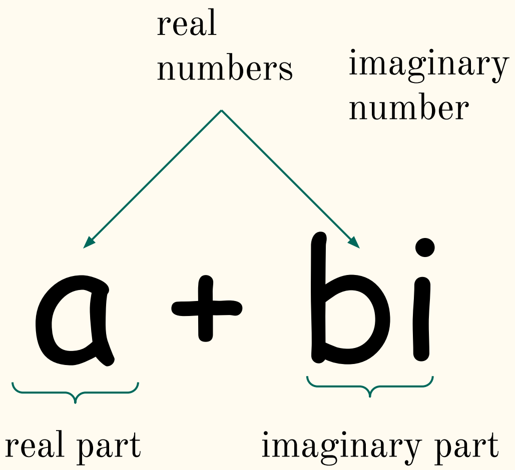
```
class Complex {
// function for evaluating equality between complex numbers

// function for evaluating inequality between complex numbers

// function for adding complex numbers

// function for inserting into output stream
---

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

// function for incrementing complex numbers

// function for converting complex number to boolean

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



Complex(5) → 5+0i
Complex(4, 8) → 4+8i
Complex(7, -3) → 7-3i

# A complex number class

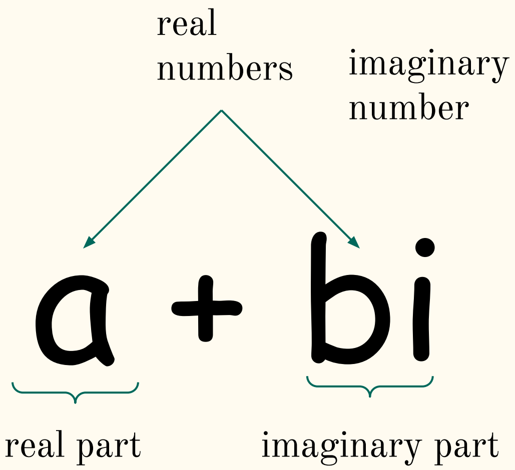
```
class Complex {
// function for evaluating equality between complex numbers

// function for evaluating inequality between complex numbers

// function for adding complex numbers

    // function for inserting into output stream
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (___) os << '+';

    }

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

// function for incrementing complex numbers

// function for converting complex number to boolean

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



Complex(5) → 5+0i
Complex(4, 8) → 4+8i
Complex(7, -3) → 7-3i

# A complex number class

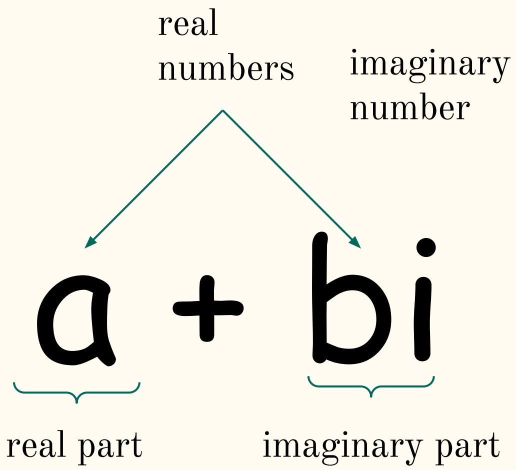
```
class Complex {
// function for evaluating equality between complex numbers

// function for evaluating inequality between complex numbers

// function for adding complex numbers

    // function for inserting into output stream
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (_2_) os << '+';

    }

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

// function for incrementing complex numbers

// function for converting complex number to boolean

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



Complex(5) → 5+0i
Complex(4, 8) → 4+8i
Complex(7, -3) → 7-3i

# Which condition (replacing blank #2) when evaluating to true will output '+'?

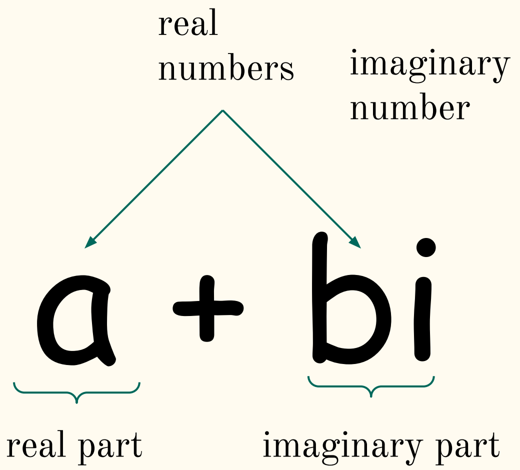
```
class Complex {
// function for evaluating equality between complex numbers

// function for evaluating inequality between complex numbers

// function for adding complex numbers

    // function for inserting into output stream
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (_2_) os << '+';

    }

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

// function for incrementing complex numbers

// function for converting complex number to boolean

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



Complex(5) → 5+0i
Complex(4, 8) → 4+8i
Complex(7, -3) → 7-3i

# A complex number class

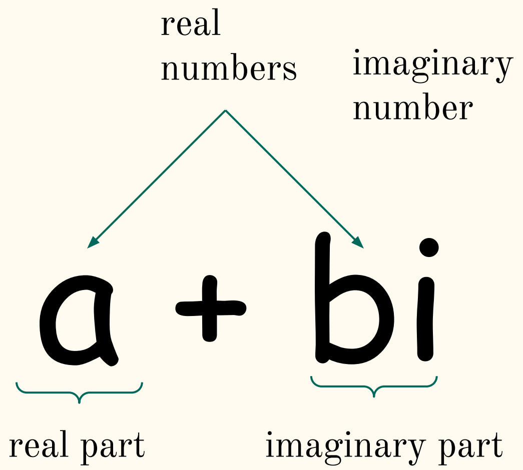
```
class Complex {
// function for evaluating equality between complex numbers

// function for evaluating inequality between complex numbers

// function for adding complex numbers

    // function for inserting into output stream
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        ---
    }

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

// function for incrementing complex numbers

// function for converting complex number to boolean

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



Complex(5) → 5+0i
Complex(4, 8) → 4+8i
Complex(7, -3) → 7-3i

# A complex number class

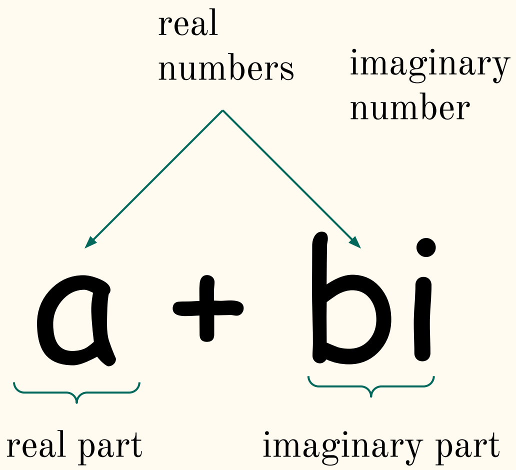
```
class Complex {
// function for evaluating equality between complex numbers

// function for evaluating inequality between complex numbers

// function for adding complex numbers

    // function for inserting into output stream
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        _3_
    }

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

// function for incrementing complex numbers

// function for converting complex number to boolean

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



Complex(5) → 5+0i
Complex(4, 8) → 4+8i
Complex(7, -3) → 7-3i

# Which statement completes the definition of the op<< function (replacing blank #3)?

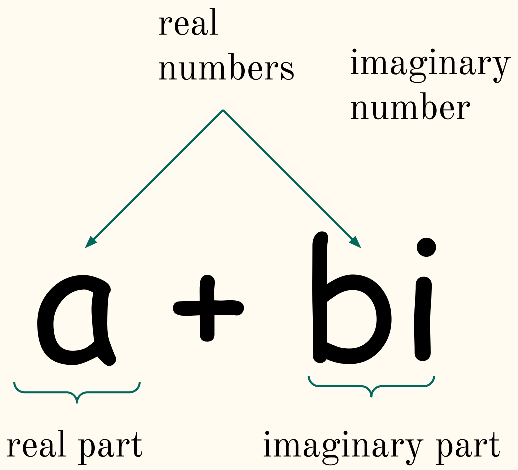
```
class Complex {
// function for evaluating equality between complex numbers

// function for evaluating inequality between complex numbers

// function for adding complex numbers

    // function for inserting into output stream
    friend ostream& operator<<(ostream& os, const Complex& rhs) {
        os << rhs.real;
        if (rhs.imag >= 0) os << '+';
        os << rhs.imag << 'i';
        _3_
    }

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

// function for incrementing complex numbers

// function for converting complex number to boolean

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



Complex(5) → 5+0i
Complex(4, 8) → 4+8i
Complex(7, -3) → 7-3i

# 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

    // function for inserting into output stream
    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

// function for converting complex number to boolean

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



Complex(5) → 5+0i
Complex(4, 8) → 4+8i
Complex(7, -3) → 7-3i

# 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) {}

// function for incrementing complex numbers

// function for converting complex number to boolean

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

# 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)
    ---

    // 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) {}

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

    // 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) {}

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

    // 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) {}

    // function for incrementing complex numbers (pre)
    _3_ operator++() { }

    // function for incrementing complex numbers (post)
    ___

// function for converting complex number to boolean

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

5+3i + 1 = 6+3i

## Which return type replaces blank #3 for the pre-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) {}

    // function for incrementing complex numbers (pre)
    _3_ operator++() { }

    // 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) {}

    // function for incrementing complex numbers (pre)
    Complex& operator++() { }

    // 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) {}

    // function for incrementing complex numbers (pre)
    Complex& operator++() {
       ---
    }

    // 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) {}

    // function for incrementing complex numbers (pre)
    Complex& operator++() {
       _4_
    }

    // function for incrementing complex numbers (post)
    ___

// function for converting complex number to boolean

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

5+3i + 1 = 6+3i

## Which expression will result in incrementing the real part of the complex number?

```
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++() {
       _4_
    }

    // 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) {}

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

    // 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) {}

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

// 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) {}

    // function for incrementing complex numbers (pre)
    Complex& operator++() {
       ++real;
       _5_
    }

// function for incrementing complex numbers (post)
___

// function for converting complex number to boolean

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

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

## Which statement (replacing blank #5) returns 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) {}

    // function for incrementing complex numbers (pre)
    Complex& operator++() {
       ++real;
       _5_
    }

// 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) {}

    // 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