Operator Overloading

Reasons

- To make your classes behave like built-in types.
  - Example: Point p1, p2; then you can overload the operator+, such that you can directly add one point object to another, such as p3 = p1 + p2, instead of using methods like p3 = p1.add(p2)
- To gain greater control over the behavior in your program
  - Example: to display the content of your own class object, by overloading the insertion operator <<; instead of using a method like cout << p1.display() << endl; you could directly cout << p1 << endl.

Limitations

- Refine only existing operators, not creating a new operator
- Certain operators cannot be redefined such as member access for object (. .*), scope operator (::) and conditional operator (?:)
- Must follow the default precedence & associativity of the operators
- Cannot redefine operators for built-in types such as + for int

Method or Global Function

- Method Based such as operator=
- Global Based where the left-hand side of the operator of different type than your own class such as operator<<
- Either, prefer to method based

Return Type

- As C++ doesn’t determine overload resolution by return type, it implies it’s up to your own implementation.

```
int kk(int i) {}

double kk(int i) {}
```

```
double kk(int i)
cannot overload functions distinguished by return type alone C/C++(311)
View Problem    Quick Fix... (⌘.)
kk(int i) {}
```

Return Type - choice

- Return the same types as the operators do for the built-in types
- Comparison operator should return bool type
- Arithmetic operator should return the object representing the result
- Operator= should return reference to the object to support chained assignment
- Operator<< should return the stream reference to support chained stream.
  - cout << a << b << c << endl;
  - operator<<(operator<<(operator<<(operator<<(cout, a), b), c), endl);

Return Type – increment (++), decrement (--)

- Prefix forms (++a, --a), the return value equals to the end value of the operand (a), in this case, it should return the reference to the object
- Postfix such as a++, a--, the return value are different from the end value of the operand (a), so it MUST not return the reference.

```
int i {1};
auto a = ++i;   // i = i+1; a = i;
auto b = i++;   // b = i; i = i+1;
cout << a << "," << b << "," << i << endl;
```

```
class cell {

    public:
        cell();
        ~cell();

        cell(int num);

        int getvalue() const;

        cell operator+(const cell & c) const;

        cell operator++();          // prefix
        cell operator++(int i);     // postfix

    friend ostream & operator<<(ostream & os, const cell & c);

    private:
        int x {0};
};
```

```
// prefix form
cell cell::operator++() {
    this->x += 1;
    return *this;
}

// postfix form
cell cell::operator++(int i) {
    auto tmp = *this;
    operator++();
    return tmp;
}
```

```cpp
ostream & operator<<(ostream & os, const cell & c) {
    os << c.x;
    return os;
}
```

```cpp
// prefix form
cell cell::operator++() {
    this->x += 1;
    return *this;
}

// postfix form
cell cell::operator++(int i) {
    auto tmp = *this;
    operator++();
    return tmp;
}
```

```cpp
cell one {1};
cout << one << endl;
auto two = ++(++one);
cout << two << endl;
cout << one << endl;
```

1
3
2

```cpp
cell one {1};
cout << one << endl;
auto two = (one++)++;
cout << two << endl;
cout << one << endl;
```

1
1
2

```cpp
cell one {1};
cout << one << endl;
auto two = one++;
cout << two << endl;
one++;
cout << one << endl;
```

1
1
3