CS 271
October 24, 2017

"this" is a pointer to the calling object

We can use "this" inside function by using the arrow operator -> or by dereferencing the pointer, then using the dot operator.

Complex class

```
Complex Complex::operator+ ( const Complex& phil )  const {

      float realPart = this -> getReal() + phil.getReal();
      float imagPart = this -> getImag() + phil.getImag();

      Complex answer( realPart, imagPart );

      return answer;
}
OR
      float realPart = (*this).getReal() + phil.getReal();
      float imagPart = (*this).getImag() + phil.getImag();

OR  (best way - don't use this at all)

      float realPart = getReal() + phil.getReal();
      float imagPart = getImag() + phil.getImag();
```

Prefix and Postfix Increment

To overload the prefix or postfix increment operator (++) we have to write a function with the name **operator++**. To distinguish the two functions, the compiler requires some way to determine which function is supposed to be used. The C++ developers decided to give the postfix increment a dummy int parameter.

Note: The compiler requires that every function have a unique **signature** (function name + parameter list).

Prototype for prefix increment:

```
      Date & operator++ ( );
```

Prototype for postfix increment:

```
      Date & operator++ ( int );
```

First, let's look at the helpIncrement in the Date class. This function changes the Date object to the next day.

```
void Date::helpIncrement( ) {

    if ( !endOfMonth( day ) )
         ++day;

    else {
         if ( month < 12 )  {
            ++month;
            day = 1;
         }
         else {  // last day of year
              day = 1;
              month = 1;
              ++year;
         }
  }
```

Now, let's look at what happens in the endOfMonth function:

```
bool Date::endOfMonth ( int testDay ) const {

     if ( month == 2 && leapYear( year ) )
        return testDay == 29;

    else
        return testDay == days[ month ];
}
```

The days array is declared at the top of the file as a const array.

```
const int Date::days[ ] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31};
```

Now we can look at the two operator functions:

```
// prefix increment

Date & Date::operator++ ( ) {  // called with ++today

    helpIncrement( );
    return *this;  // returns the calling object
}

// postfix increment

Date  Date::operator++ ( int ) {  // called with thanksgiving++

     Date temp = *this;   //  performs memberwise assignment

     helpIncrement();    // increments the calling object
     return temp;
}
```

Refresher: Here's how the ++ works for integers:

```
int x = 3;
cout << ++x << endl;   // the value of x is now 4 and it prints 4

int y = 11;
cout << y++ << endl;  // prints 11, and the value of y is now 12
```

The same thing happens when we use ++ on objects that have the operator overloaded.

```
Date today( 10, 24, 2017 );
cout << ++today << endl;   // today now contains 10, 25, 2017 and
                           // it prints October 25, 2017
                           // The printing format is determined by the
                           // code in the overloaded << operator.

Date thanksgiving( 11, 23, 2017 );
cout << thanksgiving++ << endl;  // prints November 23, 2017
                                 // and thanksgiving contains 11, 24, 2017
```

Good programming practice

Don't use ++ either prefix or postfix inside a larger statement.