COMSC-200
Lab 4

Ryan Jacoby

20 September 2020

1 Complex

```h
// Fig. 10.14: Complex.h
// Complex class definition.
#ifndef COMPLEX_H
#define COMPLEX_H

#include <iostream>
using namespace std;

class Complex {
public:
    explicit Complex( double = 0.0, double = 0.0 ); // constructor
    Complex operator+( const Complex & ) const; // addition
    Complex operator-( const Complex & ) const; // subtraction
    Complex operator*( const Complex & ) const; // multiplication
    bool operator==( const Complex & ) const;   // equals
    bool operator!=( const Complex & ) const;   // inequals
    // TODO: Implement << and >>.

    int getReal() { return real; }
    int getImaginary() { return imaginary; }
private:
    double real; // real part
    double imaginary; // imaginary part
}; // end class Complex

ostream& operator<<(ostream& out, Complex &c);
istream& operator>>(istream& in, Complex &c);

#endif
```

Listing 1: Complex.h

```cpp
// Fig. 10.15: Complex.cpp
// Complex class member-function definitions.
#include <iostream>
#include "Complex.h"  // Complex class definition
using namespace std;

// Constructor
Complex::Complex( double realPart, double imaginaryPart ) : real( realPart ),
    imaginary( imaginaryPart ) {
```

```
}

// addition operator
Complex Complex::operator+( const Complex &operand2 ) const {
    return Complex( real + operand2.real, imaginary + operand2.imaginary );
}

// subtraction operator
Complex Complex::operator-( const Complex &operand2 ) const {
    return Complex( real - operand2.real, imaginary - operand2.imaginary );
}

// multiplication operator
Complex Complex::operator*( const Complex &operand2) const {
    return Complex( (real * operand2.real) + (imaginary * operand2.imaginary), (
    real * operand2.imaginary) + (imaginary * operand2.real));
}

// equals operator
bool Complex::operator==( const Complex &operand2 ) const {
    return (real - operand2.real == 0) && (imaginary - operand2.imaginary == 0);
}

//inequals operator
bool Complex::operator!=( const Complex &operand2 ) const {
    return (real - operand2.real != 0) || (imaginary - operand2.imaginary != 0);
}

ostream& operator<<(ostream& out, Complex &c) {
    out << '(' << c.getReal() << ", " << c.getImaginary() << ')';
    return out;
}

istream& operator>>(istream& in, Complex &c) {
    int real, imaginary;
    char seperator;

    in >> real;
    in.get(seperator);
    in >> imaginary;

    c = Complex(real, imaginary);

    return in;
}
```

Listing 2: Complex.cpp

```
// Fig. 10.16: fig10_16.cpp
// Complex class test program.
#include <iostream>
#include "Complex.h"
using namespace std;

int main() {
    Complex x;
    Complex y( 4.3, 8.2 );
    Complex z( 3.3, 1.1 );

```

```
    cout << "x: " << x;
    cout << "\ny: " << y;
    cout << "\nz: " << z;

    x = y + z;
    cout << "\n\nx = y + z:\n";
    cout << x << " = " << y << " + " << z;

    x = y - z;
    cout << "\n\nx = y - z:\n";
    cout << x << " = " << y << " - " << z;

    x = y * z;
    cout << "\n\nx = y * z:\n";
    cout << x << " = " << y << " * " << z;

    if(y == z)
        cout << "\n\ny == z: True\n";
    else
        cout << "\n\ny == z: False\n";

    if(y != z)
        cout << "\ny != z: True\n";
    else
        cout << "\ny != z: False\n";

    return 0;
}
```

Listing 3: fig10_16.cpp (modified)

2 Car

```
// Ryan Jacoby

#ifndef Car_H
#define Car_H

class Car {
private:
    int speed, tank;
public:
    explicit Car(int = 0, int = 0);
    int operator+( const Car & ) const;
    int operator-( const Car & ) const;
    bool operator<( const Car & ) const;
    bool operator>( const Car & ) const;
    bool operator==( const Car & ) const;
    bool operator!=( const Car & ) const;

    int operator++();
    int operator++( int );
    int operator--();
    int operator--( int );

    int getSpeed() { return speed; }
    int getTank() { return tank; }
```

```
};

int operator+( const Car &, int );
int operator+( int, const Car & );

ostream& operator<<(ostream& out, Complex &c);
istream& operator>>(istream& in, Complex &c);

#endif
```

Listing 4: Car.h

```
// Ryan Jacoby

#include<cmath>
#include<sstream>
#include "Car.h"

using namespace std;

Car::Car(int tank, int speed) {
    this->tank = tank;
    this->speed = speed;
}

int Car::operator+( const Car c& ) {
    return tank + c.tank;
}

int Car::operator-( const Car c& ) {
    return abs(tank - c.tank);
}

bool Car::operator<( const Car c& ) {
    return tank - c.tank < 0;
}

bool Car::operator>( const Car c& ) {
    return tank - c.tank > 0;
}

bool Car::operator==( const Car & ) {
    return tank - c.tank == 0;
}

bool Car::operator!=( const Car & ) {
    return tank - c.tank != 0;
}

int Car::operator++() {
    speed++;
    return speed;
}

int Car::operator++( int ) {
    speed++;
    return speed - 1;
}

```

```
int Car::operator--() {
    if(speed > 0) {
        speed--;
        return speed;
    }

    return speed;
}
int Car::operator--( int ) {
    if(speed > 0) {
        speed--;
        return speed + 1;
    }

    return speed;
}


int operator+( const Car c&, int n ) {
    return c.getTank() + n;
}
int operator+( int n, const Car c& ) {
    return c.getTank() + n;
}

ostream& operator<<(ostream& out, Complex &c) {
    out << "Tank: " << c.getTank() << "\tSpeed: " << c.getSpeed();
    return out;
}
istream& operator>>(istream& in, Complex &c) {
    int tank, speed;
    char seperator;

    in >> tank;
    in.get(seperator);
    in >> speed;

    c = Car(tank, speed);

    return in;
}
```

Listing 5: Car.cpp