Chapter 9

Implementing operators using other operators

```
#include <iostream>

class Point3D {
 private:
  int x_, y_, z_; // Private data members

 public:
  Point(int x = 0, int y = 0, int z = 0); // Constructor
  int X() const; // Getters
  int Y() const;
  int Z() const;
  void SetX(int x); // Setters
  void SetY(int y);
  void SetZ(int z);
  void Print() const;
}

int main() {
  Point3D p0(2, 5, 3);
  Point3D p1(1, 0, 2);

  Point3D p2(p0.X() + p1.X(), p0.Y() + p1.Y(), p0.Z() + p1.Z());
  p2.Print();

  return 0;
}

// Constructor - The default values are specified in the declaration
Point::Point(int x, int y, int z) : x_(x), y_(y), z_(z) { } // Using initializer
list

// Getters
int Point::X() const { return x_; }
int Point::Y() const { return y_; }
int Point::Z() const { return z_; }

// Setters
void Point::SetX(int x) { this->x_ = x; }
void Point::SetY(int y) { this->y_ = y; }
void Point::SetZ(int z) { this->z_ = z; }

// Public Functions
void Point::Print() const {
  cout << "(" << x << "," << y << "," << z << ")" << endl;
}
```

```
#include <iostream>

int main() {
  Point3D p0(2, 5, 3);
  Point3D p1(1, 0, 2);

  Point3D p2 = p0 + p1;
  std::cout << p2 << std::endl;

  return 0;
}
```

operator overloading