Introduction

This document provides a basic introduction to C++ Operator Overloading. It should be used in conjunction with the code files for the C++ example class Point which demonstrates the basics of operator overloading using a class to represent an x-y coordinate pair. For any given topic discussed in this document, there are many additional important details not discussed because this is a basic introduction. The official online resources for C++ can be consulted for these details. Links to some of these resources, specifically Microsoft and C++ Reference, are provided at the end in the **Resources** section. Both are excellent resources, and the Microsoft links are more beginner friendly and easier to understand.

Note: for the purposes of demonstration, the std namespace is being used by including the statement using namespace std. This is just to make code easier to read. For example, it allows string to be explicitly written instead of std::string to refer to a C++ string, and it allows *cout* to be explicitly written instead of std::*cout*. It is okay to use the std namespace for the purposes of learning, teaching, or demonstration, but in real C++ code it is not a good practice.

Table of Contents

Topics	**Page**
Operator Overloading	2
Resources	3

Operator Overloading

The reason for operator overloading is it allows for classes to be used in intuitive ways with built in C++ operators. Some of the most commonly used C++ operators are:

addition, subtraction, and unary minus	+, -, and -
multiplication and division	* and /
equivalent and not equivalent	== and !=
pre and post increment	++ and ++
pre and post decrement	-- and --
output stream/insertion	<<, typically cout
input stream/extraction	>>, typically cin

As an example, take the following class called Point which stores the x-y coordinate pair of a point.

```
class Point {
public:
    // ….
    // ….
private:
    double x, y;
};
```

There are many ways in which Point could be intuitively used with C++ operators. For example, to print out a coordinate pair, it would make sense to do….

```
Point p(3, 4);
cout << p;
```

….which would print the following to the console:

```
(3, 4)
```

Without overloading the << operator for the Point class, this is invalid C++ code. In order to print out that statement, the following would need to be done:

```
cout << "(" << p.getX( ) << ", " << p.getY( ) << ")";
```

As another example, it would make sense to want to calculate the sum of two x-y coordinate pairs and store them in a Point object such as….

```
Point p1;
Point p2(3, 4);
Point p3(-2, 5);
p1 = p2 + p3;
```

….which would result in p1 having the x-y coordinate pair

```
(1, 9)
```

Without overloading the + operator for the Point class, this is invalid C++ code *(The = operator overload (aka copy assignment operator) is also used in the above example. Technically, the compiler copy assignment operator is sufficient for this since x and y are not allocated with new. However, it is always poor practice to rely on the compiler generated copy assignment operator and a user-defined copy assignment operator should be written).* In order for (1, 9) to be stored in p1, the following would need to be done:

```
p1 = Point(p2.getX( ) + p3.getX( ), p2.getY( ) + p3.getY( ));
```

Resources

Classes

- Microsoft: https://docs.microsoft.com/en-us/cpp/cpp/operator-overloading?view=msvc-160
- C++ Reference: https://en.cppreference.com/w/cpp/language/operators