Introduction

This document provides a basic introduction to C++ Operator Overloading. It should be used in conjunction with the code file OperatorOverloadingBasics.cpp which gives an example of operator overloading using a class named Point to represent an x-y coordinate pair

For any given topic discussed in regards to operator overloading, there are additional important details not discussed because this is a basic introduction. There are numerous official online resources for C++ that can be consulted for these details. Links to some of these resources, specifically *Microsoft*, *C++ Reference*, and the I*SO C++ Standard*, are provided at the end in the **Resources** section.

- Microsoft and C++ Reference provide a general reference for any C++ related topic. Microsoft's website is a bit easier to understand and more beginner friendly.
- The ISO C++ Standard contains the C++ Core Guidelines which is a discussion on the current C++ best practices and standards. It is not meant for learning about a C++ topic but rather meant for someone who already knows about a specific C++ topic, and is looking for a further discussion on best practices.

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

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. For this particular class, the compiler-generated copy assignment operator is ok since the class doesn't manage any resources).* 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

Operator Overloading

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

ISO C++ Core Guidelines

Main Website: https://isocpp.org/
Core Guidelines: http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines