ESE 224 – Advanced Programming and Data Structures

Fall 20, Xin Wang

Laboratory 02: Operator Overloading and Output Format

This laboratory is to be performed the week starting Sep. 14th.

Prerequisite Reading

Lecture Slides for Chapter 2 and Chapter 3

Lecture Content Review

I. Operator Overloading in Class Definition

- Provide a new definition of an existing operator to work with the class
- Format:
  - Declaration:

```
returnType operator operatorName (parameters);
```

    e.g. double operator – (const Point& p2) const;
    void operator * (int n);

  - Implementation:

```
ReturnType className::operator operatorName (parameters)
{
        Implementation;
}
```

    e.g. double Point::operator – (const Point& p2) const {…}
    void Circle::operator * (int n) {…}

  - Usage:

```
objectName operatorName parameters
```

    e.g. p1 – p2;
    c1 * 2;

II. Standard Input&Output

- Need to include library: #include <iostream>
- Input – cin
  Usage:
  - cin >> variable; → all types of variables, skip all whitespace characters
  - cin.get(variable); → only for character type variable
- Output – cout
  Usage:
  - cout << variable << endl;
- Format Control:
  - Using member function and format flags:
    - E.g. cout.setf(flag), cout.unsetf(), cout.precision(4) before cout << variable << endl;

Flag	Meaning
ios::showpoint	display the decimal point
ios::fixed	fixed decimal notation
ios::scientific	scientific notation
ios::right	right justification
ios::left	left justification

  - Using manipulators (needs library: #include <iomanip>):
    - E.g. cout << setprecision(4) << variable << endl;

Manipulator	Meaning
showpoint	display the decimal point
fixed	fixed decimal notation
scientific	scientific notation
setprecision(n)	set the number of significant digits to be printed to the integer value n
setw(n)	set the minimum number of columns for printing the next value to the integer value n
right	right justification
left	left justification

Purpose

The purpose of this laboratory is to practice using operator overloading in the class definition. Also, try to restrict the output format in different ways.

Laboratory Tasks

1. **Point Class: Last time, you have practiced defining Point Class on your own. This time, you're required to add some more member functions to this class:**

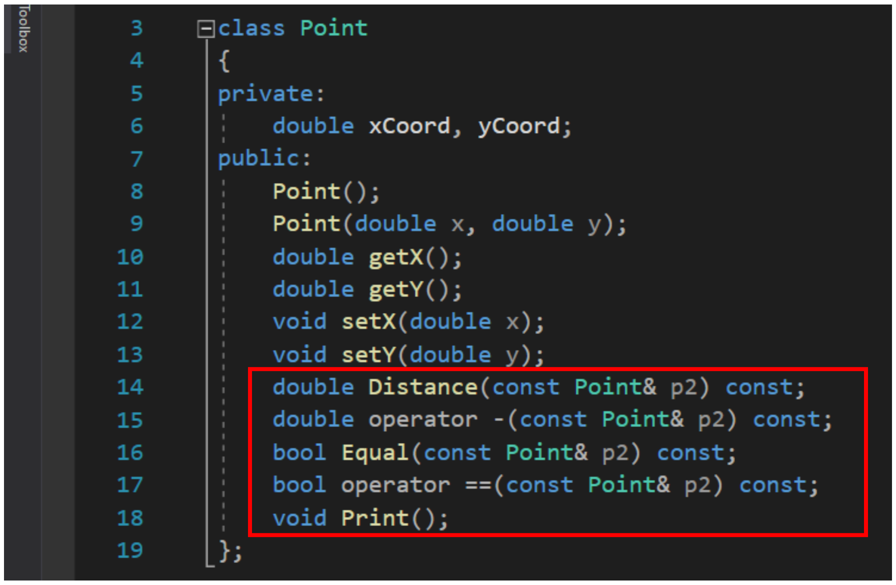
- **Distance(): A function to calculate the distance between two points;**
- **Overloading an operator - to calculate the distance between two points;**
- **Equal(): A function to judge whether two points are the same, if they are the same, the answer is true (1), otherwise the answer is false (0) ;**
- **Overloading an operator == to see whether two points are the same;**

- **Print(): A function to print all the information of the point (xCoord and yCoord), all the numbers should only keep 3 digits to the right of the decimal point.**

Similarly, you need to test all these functions in the main file.

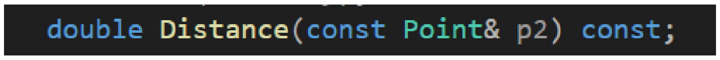
"Point.h"

Let's first see the class declaration, based on what we wrote last time, we need to add some more member functions as shown below:



```cpp
class Point
{
private:
    double xCoord, yCoord;
public:
    Point();
    Point(double x, double y);
    double getX();
    double getY();
    void setX(double x);
    void setY(double y);
    double Distance(const Point& p2) const;
    double operator -(const Point& p2) const;
    bool Equal(const Point& p2) const;
    bool operator ==(const Point& p2) const;
    void Print();
};
```

Be careful, we always put a keyword "operator" before the operator we want to overload. As for the keyword "const", it means we won't change the data members in this function. E.g. in this function

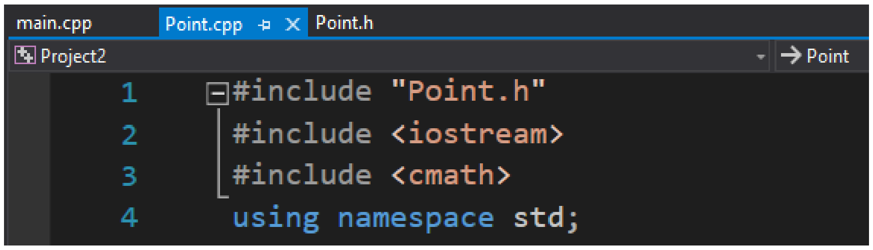


```cpp
double Distance(const Point& p2) const;
```

the second "const" indicates that we won't change the xCoord and yCoord while the first "const" means we won't change the xCoord and yCoord of the input parameter p2. Here, we have a symbol "&" after "Point", it represents a kind of method of passing parameters, we will talk about it in later chapters.

"Point.cpp"

Just to emphasize, before implementing these functions, don't forget to include this header file in the first line. Also, since we will use some pre-defined functions to calculate distances and cout function to print

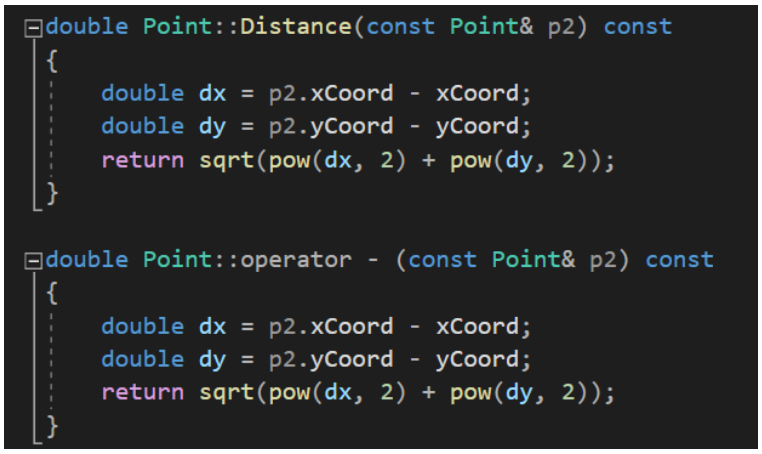


```cpp
#include "Point.h"
#include <iostream>
#include <cmath>
using namespace std;
```

the information, we also need two more libraries <iostream> and <cmath>.

Let’s take a look at these new added functions one-by-one.

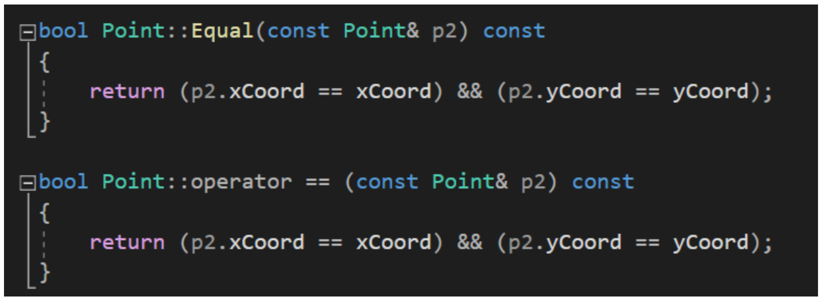
(1) Distance() and overloaded operator - :
From the code, we can see that these two functions actually do the same thing. We need first calculate dx and dy, then use the pre-defined functions sqrt() and pow() in <cmath>. pow(m, n) calculates the m to the power of n, and sqrt(k) calculates the squared root of k.



```
double Point::Distance(const Point& p2) const
{
    double dx = p2.xCoord - xCoord;
    double dy = p2.yCoord - yCoord;
    return sqrt(pow(dx, 2) + pow(dy, 2));
}

double Point::operator - (const Point& p2) const
{
    double dx = p2.xCoord - xCoord;
    double dy = p2.yCoord - yCoord;
    return sqrt(pow(dx, 2) + pow(dy, 2));
}
```

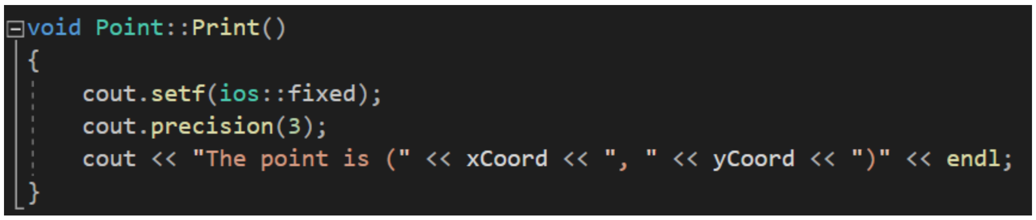
For the operator overloading, the keyword “Point::” is put before “operator”, or you can regard “operator -” as a function name here.

(2) Equal() and overloaded operator == :
Similarly, these two functions are used to determine whether the x-coordinate and y-coordinate are separately equal to the corresponding coordinate of input parameter. The implementations of them are the same, but we will see that the usage of these two functions are different in the main file.



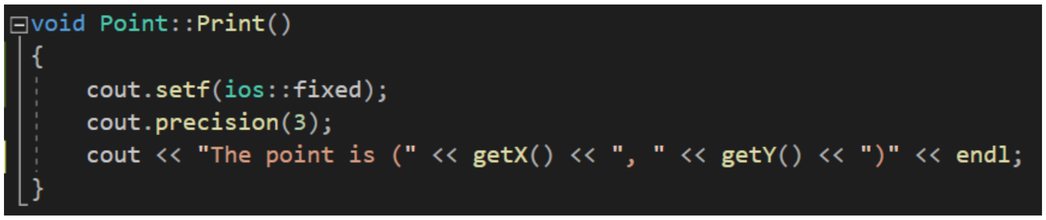
```
bool Point::Equal(const Point& p2) const
{
    return (p2.xCoord == xCoord) && (p2.yCoord == yCoord);
}

bool Point::operator == (const Point& p2) const
{
    return (p2.xCoord == xCoord) && (p2.yCoord == yCoord);
}
```

(3) Print():
In order to use the standard output function, we need the library <iostream>, and we can restrict the output format arbitrarily. In this problem, we need 3 digits kept after the decimal point, so we use cout.setf(ios::fixed) and cout.precision(3) here. If you want to use manipulators, you need also include the library <iomanip>.



```cpp
void Point::Print()
{
    cout.setf(ios::fixed);
    cout.precision(3);
    cout << "The point is (" << xCoord << ", " << yCoord << ")" << endl;
}
```

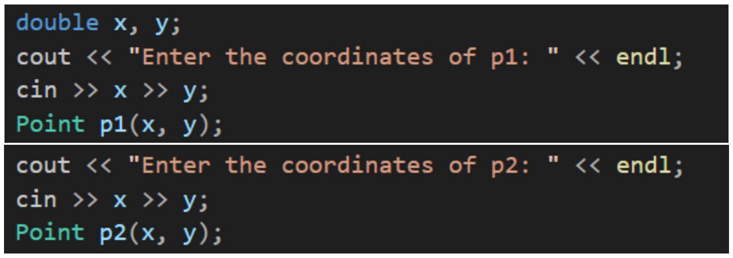
Besides directly using xCoord and yCoord, we can also call other member functions like this:



```cpp
void Point::Print()
{
    cout.setf(ios::fixed);
    cout.precision(3);
    cout << "The point is (" << getX() << ", " << getY() << ")" << endl;
}
```

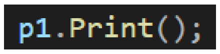
"main.cpp"

In the main file, don't forget to include the class header file and other necessary libraries at the beginning, then we create two objects of Point Class:



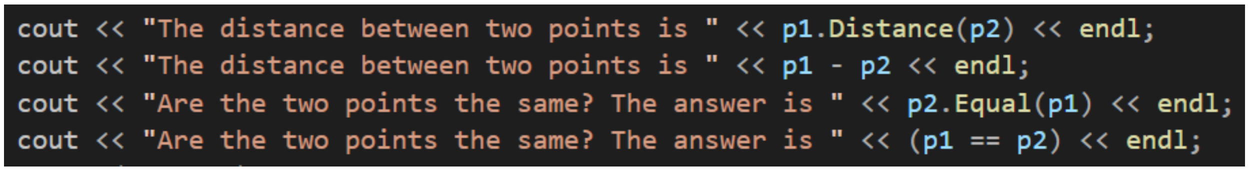
```cpp
double x, y;
cout << "Enter the coordinates of p1: " << endl;
cin >> x >> y;
Point p1(x, y);
cout << "Enter the coordinates of p2: " << endl;
cin >> x >> y;
Point p2(x, y);
```

The usage of Print() function is very easy:



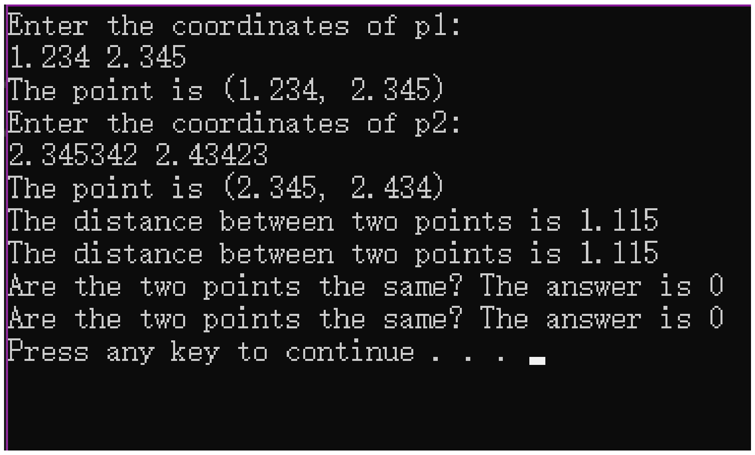
```cpp
p1.Print();
```

Now we want to compare the usage of general functions Distance() and Equal() with the overloaded operators - and ==. Just like Print(), when using general functions, we use the dot operation, but for operators, we follow the common usage format as usual.



```cpp
cout << "The distance between two points is " << p1.Distance(p2) << endl;
cout << "The distance between two points is " << p1 - p2 << endl;
cout << "Are the two points the same? The answer is " << p2.Equal(p1) << endl;
cout << "Are the two points the same? The answer is " << (p1 == p2) << endl;
```

Although the usage is not the same, but actually they do the same thing, if you run the code, you will see that the general member functions and the overloaded operators have the same output.



```
Enter the coordinates of p1:
1.234 2.345
The point is (1.234, 2.345)
Enter the coordinates of p2:
2.345342 2.43423
The point is (2.345, 2.434)
The distance between two points is 1.115
The distance between two points is 1.115
Are the two points the same? The answer is 0
Are the two points the same? The answer is 0
Press any key to continue . . .
```

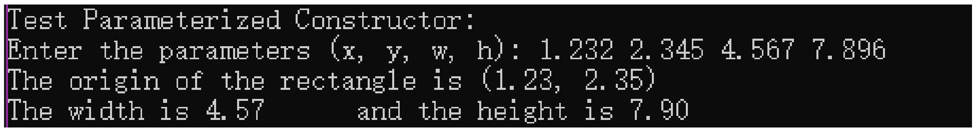
2. Point Class: Still in the Point Class, this time you need to add two overloaded operators:

- *** : used to multiply the point with a double number n, as a result, the x-coordinate and y-coordinate should become n times as before;**
- **> : used to compare two points, return true (1) if the distance between the current point and the Origin (0, 0) is larger than the one between the parameter point and the Origin, e.g. p1(2, 0), p2(1, 0), the answer of p1 > p2 should be 1;**

You can write the code based on Problem 1, and test all your newly written operators in the main file.

3. Rectangle Class: Based on the Rectangle Class we implemented before, add three more functions:

- **Overloaded operator > : return true (1) if the area of the current rectangle is larger than the one of the parameter rectangle;**
- **Overloaded operator - : calculate the difference in areas of two rectangles;**
- **Print function: print all the attributes of the rectangle in the following format, 2 digits are kept after the decimal point and the display column for width and height should be 10 and left alignment is required.**



```
Test Parameterized Constructor:
Enter the parameters (x, y, w, h): 1.232 2.345 4.567 7.896
The origin of the rectangle is (1.23, 2.35)
The width is 4.57      and the height is 7.90
```

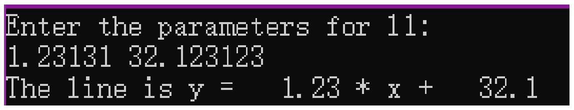
After that, test the functions in the main file.

4. Line Class: A straight line on a 2D plane can be defined as

y = k * x + b

in other words, it can be determined by two variables k (slope) and b (intercept). You’re supposed to write a class for such a line, and the member functions should include:

- **Constructors;**
- **Print function which has the following display format;**



```
Enter the parameters for l1:
1.23131 32.123123
The line is y =    1.23 * x +    32.1
```

- **Overloaded operator == : determine whether two lines are the same, including k and b are respectively equal;**
- **Overloaded operator * : multiply the slope with the given number;**
- **Overloaded operator + : add a number to the intercept;**

After defining the class, test your class in a main function by using all the member functions.