CMPT 135: Lab Work Week 3

Reference Variables, Constant Modifiers, and the this pointer

1. Consider the following C++ class:

```
class A
{
private:
    int v;
public:
    A()
    {
        setValue(0);
    }
    A(const int &v)
    {
        setValue(v);
    }
    int getValue() const
    {
        return v;
    }
    void setValue(const int &v)
    {
        v = v;
    }
    A subtract (const A &a) const
    {
        return A(getValue() - a.getValue());
    }
};
```

   a. Is this class definition valid or invalid?
   b. If it is invalid, state the error clearly and correct the error so that the class definition is valid.
   c. Once you correct the class definition, analyze the following testing main program and give the output as it would appear on the output screen.

```
int main()
{
        A a1, a2(8), a3(2);
        cout << a1.getValue() << ", " << a2.getValue() << ", " << a3.getValue() << endl;
        cout << a1.subtract(a2).getValue() << endl;
        cout << a3.subtract(a2).getValue() << endl;
        return 0;
}
```

2. Declare and define a member function named **`isNotEqual`** for the **`RationalNumber`** class that was discussed in the lecture. Write an appropriate test main program to test your member function.

3. Declare and define a member function named **`isGreaterThan`** for the **`RationalNumber`** class that was discussed in the lecture. Write an appropriate test main program to test your member function.

4. Declare and define a member function named **`isLessThan`** for the **`RationalNumber`** class that was discussed in the lecture. Write an appropriate test main program to test your member function.

5. Declare and define a member function named **`isGreaterOrEqual`** for the **`RationalNumber`** class that was discussed in the lecture. Write an appropriate test main program to test your member function.

6. Declare and define a member function named **`isLessOrEqual`** for the **`RationalNumber`** class that was discussed in the lecture. Write an appropriate test main program to test your member function.

7. Write a C++ program that creates a dynamic array of **RationalNumber** objects of user desired size, set the numerator and denominator of each element of the array to some random integer in the range [-5, 5] and finally compute and print the minimum and maximum elements of the array.

8. Write a C++ program that creates a dynamic array of **RationalNumber** objects of user desired size, set the numerator and denominator of each element of the array to some random integer in the range [-5, 5] and finally compute and print the minimum and maximum positive valued elements of the array. If there is no any positive **RationalNumber** object in the array then your program must instead print the message "No minimum or maximum positive elements".

9. In Linear algebra, a vector in 2D is an object with a magnitude and direction and it is represented by a directed straight line from the origin to a point in the 2D plane. Essentially, a vector is described by a Point in a 2D plane. Therefore one way to design a class that represents vectors in 2D is to design the class with only one member variable of type Point.

Write a C++ class named Vector2D that represents vectors in 2D space. Have proper constructors, getters, setters, and any additional member functions.

Use the following program to test your class design

```
int main()
{
	const Vector2D v1;
	Vector2D v2(Point(3, -4));
	cout << "v1 is " << v1 << ", its length is " << v1.length() << endl;
	cout << "v2 is " << v2 << ", its length is " << v2.length() << endl;
	cout << "The end point of v1 is " << v1.getEndPoint() << endl;
	v2.setEndPoint(Point(rand() % 21 - 10, rand() % 11 - 5));
	cout << "The end point of v2 is " << v2.getEndPoint() << endl;

	system("Pause");
	return 0;
}
```

Sample Run Output

```
v1 is (0, 0)--->(0, 0), its length is 0
v2 is (0, 0)--->(3, -4), its length is 5
The end point of v1 is (0, 0)
The end point of v2 is (-2, 3)
Press any key to continue . . .
```

CMPT 135: Lab Work Week 4

Operator overloading, friend functions, and static members

10. Extend the **RationalNumber** class discussed in the lecture by adding the following overloaded operators:

- Binary **subtraction** operator that performs r1 - r2
- Binary **subtraction** operator that performs r1 - **integer**
- Binary **subtraction** operator that performs **integer** - r2
- Binary **multiplication** operator that performs r1 * r2

- Binary **multiplication** operator that performs r1 * **integer**
- Binary **multiplication** operator that performs **integer** * r2
- Binary **division** operator that performs r1 / r2
- Binary **division** operator that performs r1 / **integer**
- Binary **division** operator that performs **integer** / r2
- Binary **compound addition** operator that performs r1 += r2
- Binary **compound addition** operator that performs r1 += **integer**
- Binary **compound subtraction** operator that performs r1 -= r2
- Binary **compound subtraction** operator that performs r1 -= **integer**
- Binary **compound multiplication** operator that performs r1 *= r2
- Binary **compound multiplication** operator that performs r1 *= **integer**
- Binary **compound division** operator that performs r1 /= r2
- Binary **compound division** operator that performs r1 /= **integer**
- Binary **equal** operator that performs r1 == r2
- Binary **equal** operator that performs r1 == **integer**
- Binary **equal** operator that performs **integer** == r2
- Binary **notequal** operator that performs r1 != r2
- Binary **notequal** operator that performs r1 != **integer**
- Binary **notequal** operator that performs **integer** != r2
- Binary **greaterthan** operator that performs r1 > r2
- Binary **greaterthan** operator that performs r1 >**integer**
- Binary **greaterthan** operator that performs **integer**> r2
- Binary **lessthan** operator that performs r1 < r2
- Binary **lessthan** operator that performs r1 <**integer**
- Binary **lessthan** operator that performs **integer**< r2
- Binary **greaterorequal** operator that performs r1 >= r2
- Binary **greaterorequal** operator that performs r1 >= **integer**
- Binary **greaterorequal** operator that performs **integer**>= r2
- Binary **lessorequal** operator that performs r1 <= r2
- Binary **lessorequal** operator that performs r1 <= **integer**
- Binary **lessorequal** operator that performs **integer**<= r2
- Unary operator **pre decrement- -** as in **--r**

- Unary operator **post decrement --** as in **r--**

Write a test main program to test all your overloaded operators.

11. Write a non-member function named **sortArray** that takes an array of **RationalNumber** objects and sorts the array using any of the sequential sorting algorithms (insertion sort, bubble sort or selection sort).

Write a main program to test your function. In your program ask the user for the size of an array, create a dynamic array of **RationalNumber** data type with the specified size, fill the array with some random rational numbers of your choice, print the elements of the array, sort the array by calling the **sortArray** function, and finally print the sorted array to see the correctness of your sorting function.

12. Consider the following sequence of **RationalNumber** objects:

r1 = 1/3, r2 = 1/2, r3 = 5/6, r4 = 4/3, r5 = 13/6, r6 = 7/2,...

That is the first element is 1/3, the second element is 1/2 and every element afterwards is the sum of the two elements preceding it. Write a non-member and a non-friend function named **elementAt** that takes an integer argument **n** and returns the **n**th**RationalNumber** object in the sequence.

For example, the function call **elementAt(1)** must return 1/3, **elementAt(2)** must return 1/2, **elementAt(5)** must return 13/6 etc. Write a test main program to test your function.

13. Consider the sequence given in Question #12 above. Write a non-member and a non-friend function named **elementIndex** that takes a **RationalNumber** object and returns its index in the sequence if the **RationalNumber** object is found in the sequence; otherwise returns -1. For example, the function call **elementIndex(1/3)** must return 1, **elementIndex(1/2)** must return 2, **elementIndex(5/6)** must return 3, **elementIndex(7/2)** must return 6, **elementIndex(3/2)** must return -1, etc.

Write a test main program to test your function.

14. Consider a complex number object as in **3+5i** which you have learned in mathematics. A complex number is represented by a class with two double data type private member variables that store the real part and the imaginary part of the complex number. Design a class that represents a complex number. Add all necessary constructors, getters, setters, operators, and any additional member or friend functions as you see fit.

Use the following program to test your class design.

```
int main()
{
	cout << "At the beginning " << ComplexNumber::getCount() << " objects are constructed." << endl;

	ComplexNumber c1, c2(1.4, -2.5), *c3;
	cout << "So far " << ComplexNumber::getCount() << " objects are constructed." << endl;

	cout << "c1 real part is " << c1.getReal() << endl;
	cout << "c1 imaginary part is " << c1.getImaginary() << endl;

	cout << "c1 is " << c1 << endl;
	cout << "c2 is " << c2 << endl;

	c3 = new ComplexNumber;
	*c3 = c1 - c2;
	ComplexNumber c4 = c1 + c2;
	cout << "So far " << ComplexNumber::getCount() << " objects are constructed." << endl;

	cout << "*c3 is " << *c3 << endl;
```

```
        cout << "c4 is " << c4 << endl;

        ComplexNumber c5 = -c2;
        cout << "c5 is " << c5 << endl;
        cout << "So far " << ComplexNumber::getCount() << " objects are constructed." << endl;

        ++c1;
        cout << "Now c1 is " << c1 << endl;
        cout << "c2 is " << c2 << endl;
        cout << "c2++ is " << c2++ << endl;
        cout << "Now c2 is " << c2 << endl;
        cout << "So far " << ComplexNumber::getCount() << " objects are constructed." << endl;

        system("Pause");
        return 0;
}
```

Sample Run Output

```
At the beginning 0 objects are constructed.
So far 2 objects are constructed.
c1 real part is 0
c1 imaginary part is 0
c1 is 0 + 0i
c2 is 1.4 + -2.5i
So far 5 objects are constructed.
*c3 is -1.4 + 2.5i
c4 is 1.4 + -2.5i
c5 is -1.4 + 2.5i
So far 6 objects are constructed.
Now c1 is 1 + 0i
c2 is 1.4 + -2.5i
c2++ is 1.4 + -2.5i
Now c2 is 2.4 + -2.5i
So far 10 objects are constructed.
Press any key to continue . . .
```

15. In the imperial system of measuring of weight, a **Weight** is represented by two integer values representing pounds and ounces where one pound is equal to 16 ounces. Write a C++ class named **Weight** that represents weight in the imperial system. Have proper constructors, getters, setters, operators, and any additional member or friend functions. Remember for any Weight object at any time, you must keep the value of the pound greater or equal to zero and the value of the ounces between 0 and 15.

Write a test main program to test the functionality of your class.

16. Consider your Vector2D class that you have designed in Question #9 above. Add to your class operators (binary arithmetic operators, binary relational operators, unary arithmetic operators and streaming operators).

Please note that

- For binary arithmetic operators implement only vector addition and subtraction
- For binary relational operators use the length of the vector for comparison. Therefore v1 > v2 is true if and only if v1 is longer than v2; and so on so forth.
- For the unary pre/post increment operators, your overloaded function must increment the length of the calling object by 1 but not alter the direction of the object. For example, if v1 has length 3.2 then

++v1 and v1++ should modify v1 so that its length becomes 4.2 but its direction remains the same. Similarly, for the unary pre/post decrement operators.

- For the input and output streaming operators, read and print vector objects in a nice format of your choice.

Use the following program to test your class design.

```
int main()
{
	const Vector2D v1;
	Vector2D v2(Point(3, -4));
	cout << "v1 is " << v1 << ", its length is " << v1.length() << endl;
	cout << "v2 is " << v2 << ", its length is " << v2.length() << endl;
	cout << "The end point of v1 is " << v1.getEndPoint() << endl;
	v2.setEndPoint(Point(rand() % 21 - 10, rand() % 11 - 5));
	cout << "The end point of v2 is " << v2.getEndPoint() << endl;

	cout << "The sum of " << v1 << " and " << v2 << " is " << v1 + v2 << endl;
	cout << "The difference of " << v1 << " and " << v2 << " is " << v1 - v2 << endl;
	cout << "negative of " << v2 << " is " << -v2 << endl;
	cout << v1 <<  " multiplied by 5 is " << v1 * 5 << endl;
	cout << v2 <<  " divided by 5 is " << v2 / 5 << endl;

	if (v1 > v2)
		cout << v1 << " is longer than " << v2 << endl;
	else
		cout << v1 << " is not longer than " << v2 << endl;

	cout << "v2 is " << v2 << endl;
	cout << "\tIts length is " << v2.length() << endl;
	cout << "++v2 is " << ++v2 << endl;
	cout << "\tIts length is " << v2.length() << endl;

	Vector2D v3(Point(rand() % 21 - 10, rand() % 11 - 5));
	cout << "v3 is " << v3 << endl;
	cout << "\tIts length is " << v3.length() << endl;
	cout << "--v3 is " << --v3 << endl;
	cout << "\tIts length is " << v3.length() << endl;

	cout << "v2 is " << v2 << endl;
	cout << "\tIts length is " << v2.length() << endl;
	cout << "v2++ is " << v2++ << endl;
	cout << "\tIts length is " << v2.length() << endl;

	cout << "v3 is " << v3 << endl;
	cout << "\tIts length is " << v3.length() << endl;
	cout << "v3-- is " << v3-- << endl;
	cout << "\tIts length is " << v3.length() << endl;

	system("Pause");
	return 0;
}
```

Sample Run Output

```
v1 is (0, 0)--->(0, 0), its length is 0
v2 is (0, 0)--->(3, -4), its length is 5
The end point of v1 is (0, 0)
The end point of v2 is (-2, 3)
The sum of (0, 0)--->(0, 0) and (0, 0)--->(-2, 3) is (0, 0)--->(-2, 3)
The difference of (0, 0)--->(0, 0) and (0, 0)--->(-2, 3) is (0, 0)--->(2, -3)
negative of (0, 0)--->(-2, 3) is (0, 0)--->(2, -3)
```

```
(0, 0)--->(0, 0) multiplied by 5 is (0, 0)--->(0, 0)
(0, 0)--->(-2, 3) divided by 5 is (0, 0)--->(-0.4, 0.6)
(0, 0)--->(0, 0) is not longer than (0, 0)--->(-2, 3)
v2 is (0, 0)--->(-2, 3)
        Its length is 3.60555
++v2 is (0, 0)--->(-2.5547, 3.83205)
        Its length is 4.60555
v3 is (0, 0)--->(9, 4)
        Its length is 9.84886
--v3 is (0, 0)--->(8.08619, 3.59386)
        Its length is 8.84886
v2 is (0, 0)--->(-2.5547, 3.83205)
        Its length is 4.60555
v2++ is (0, 0)--->(-2.5547, 3.83205)
        Its length is 5.60555
v3 is (0, 0)--->(8.08619, 3.59386)
        Its length is 8.84886
v3-- is (0, 0)--->(8.08619, 3.59386)
        Its length is 7.84886
Press any key to continue . . .
```