Programming Assignment #4
Due: 11/13/2022 23:59:59

Topic: Generic vector class with operator overloading

Objective: build a generic geometric vector class to exercise operator overloading in C++.

Description:
You are asked to write a class called `Vector`. As you should remember from basic physics and linear algebra, a vector is represented by a set of coordinates. For example, a 2-dimensional vector is represented by an (x, y) pair and a 3-dimensional vector is represented by (x, y, z). Vectors in higher dimensions are described in a similar fashion. The mathematical nature of this data type makes it a good candidate for using operator overloading, which is the main goal of this assignment.

The Vector class will represent a set of coordinates, stored as doubles and the dimensionality of a Vector will be flexible (growable and shrinkable in run-time). The required functionality will be given below.

We are providing a `Makefile` and a main program for testing. You are asked to design this class from scratch according to the given specifications below.

You are asked to provide the following functionality for the class:

- **Constructor**: You should have a Vector constructor that takes the dimensionality of the vector as its only parameter. For example,
  ```
  Vector v1(4); // creates a 4-d vector: (0, 0, 0, 0)
  ```
  Any dimensionality greater than 0 is permissible; there should be no hard-coded limit. If no parameter is passed to the constructor (the default constructor), the dimensionality is 2 by default. For example,
  ```
  Vector v2; // create a 2-dimensional vector: (0, 0)
  ```
  All coordinates should be initialized to 0 as shown above. In addition, to facilitate testing your code, you should also implement a constructor that takes a dimensionality and an array of doubles as parameters and initialize the vector with the given array. For example,
  ```
  double darray[2]={1.0, 2.0};
  Vector v3(2, darray); // create a 2-d vector: (1.0, 2.0)
  ```
- **The "Big Three":** In C++, the term "Big Three" refers to the *destructor*, *copy constructor*, and *assignment overloading*. The rule of thumb for the Big Three: *If you write a class and it needs a destructor, or a copy constructor, or an assignment operator overloading, then it needs all three.* Since the Big Three typically deals with dynamically allocated memory, and if you need special treatment for one, it is likely that you will need special treatments for the others as well. Therefore, in the Vector class, you have to implement these Big Three.
- **`operator[]`:** You have to overload the `operator[]` so that it accesses the coordinates of a Vector (as a left or right value in an expression). `operator[]` should run in constant time. For example,
  ```
  v2[0]=1.2;
  v2[1]=3.4;
  ```
  In the case of out-of-bound index, you should print out an error message indicating the failure. In a real application, you may choose to exit the program entirely since such operation may result in an uncertain situation. However, in order to facilitate testing your code, you are asked to return 0 instead if an error occurs.

- **Accessing/Changing dimensionality:** The member function `getDimension` returns the dimensionality of a Vector. For example,

```
v2.getDimension(); // returns 2
```

The member function `setDimension` let you change the dimensionality of a Vector after it has been created. For example,

```
v2.setDimension(4); // v2: (1.2, 3.4, 0.0, 0.0)
v2.getDimension(); // returns 4
```

If the new dimension is greater than the previous one, the new coordinates are appended to the end of the old Vector and their values are all set to 0. If the new dimension is smaller, then the extra coordinates are truncated. For example,

```
v2[2]=5.6;
v2[3]=7.8; // v2 now contains (1.2, 3.4, 5.6, 7.8)
v2.setDimension(2); // v2 now contains (1.2, 3.4)
```

- **Arithmetic Operators:** You are asked to overload the following operators:
  - ✓ *Unary negation:* To negate a Vector, negate the entire coordinates of the Vector.
  - ✓ *Addition and subtraction:* To add (or subtract) two Vectors, add (or subtract) the individual coordinates of the two Vectors. In addition to the `operator+` and operator-, you also have to overload `operator+=` and `operator-=`. You should print out an error message and do no operation if the dimensions of the two Vectors are not the same.
  - ✓ *Scalar multiplication:* This is the multiplication of a double and a Vector. The result is a Vector whose coordinates have each been multiplied by the scalar. Note that you have to consider two cases: scalar*Vector and Vector*scalar. In addition, do not forget to overload `*=`.

- **Equality Operators:** Overload `operator==` and `operator!=`, the equality and inequality operators. Two Vectors are equal if their dimensionalities are the same and their corresponding coordinates are the same as well. Note that there is no unique definition for the `operator>` and `operator<` or any of the other comparison operators. Therefore, please do not overload any other comparison operators to create confusion.

- **Length:** The length member function should compute and return a double containing the length of a Vector. The length of a Vector is the square root of the sum of the squares of the Vector's coordinates. For example,

```
Vector v3(3);
v3[0]=5;
v3[1]=7;
v3[2]=9;
v3.length(); // returns the value of sqrt(5*5 + 7*7 + 9*9)
```

- **Operator<<:** Overload the global operator << such that you can nicely display a vector to an output stream. For example,

```
cout << "v2 = " << v2;
```

produces the following console output:

```
v2 = (1.2, 3.4)
```

Assignment Directory: `/usr/local/class/oop/assign/assign4`

What to Hand in: You should write the implementation of the Vector class in `Vector.cc` with the given `Vector.h` file. As usual, you need to hand in a hardcopy of your code in class, and hand in the complete code electronically. Make sure that the TA only needs to issue a 'make' to generate your executable.