CSCI 241 Assignment 4
100 points

Assignment Overview

This program creates and implements a class to manipulate 3-dimensional vectors. From math, a vector is a quantity that represents direction and magnitude. Think of an arrow that points in 3D space. A 3D vector can be represented by 3 floating point numbers, which represent how much the vector is pointing in the x, y, and z directions.

If math is not a strength for you, don't worry. This assignment is *not* about the math. It's about implementing lots of different operator overloading. The required math will be explained for each operation.

A *driver program* is provided for this assignment to test your implementation. You don't have to write the tests.

Purpose

The purpose of this assignment is to give you some experience in operator overloading.

Program

You will need to write a single class for this assignment, the `Vector3` class. You will need to implement several methods and functions associated with this class.

`class Vector3`

Data members

This class contains 3 floating point data values representing the x, y, and z components of the vector. These may be stored as data type `double` or data type `float`. There are a couple of possible ways of implementing the data members, all correct.

Methods and associated functions

- Constructor

  The class should have one constructor that takes three arguments which should set the x, y, and z vector components in that order. There should be only one constructor. Default argument values of 0 should be used for this constructor. Implementations with more than one constructor will be penalized.

- `operator<<`

  The stream insertion operator should be overloaded so that a `Vector3` can be sent to the standard output. Take a look at the provided sample output to find out how the vector should look when printed.

- `operator+`

- The addition operator should be overloaded to take two `Vector3`s and return a `Vector3`. The components of the result are computed by simply adding the components of the operands. For example (1, 2, 3) + (4, 5, 6) should have a result of (5, 7, 9). The operands should not be altered.

- `operator-`

  The binary subtraction operator should also be overloaded to take two `Vector3`s and return a `Vector3`. The result is the component-wise difference of the operands. For example (1, 2, 3) - (4, 6, 8) should have a result of (-3, -4, -5). The operands should not be altered.

- `operator*`

  The binary multiplication operator should be overloaded *three* times. The first form, called the *scalar product* takes two `Vector3`s and produces a single floating point value. The scalar product is computed by multiplying the corresponding components of the two vectors and adding the results. For example the scalar product of (1, 2, 3) and (4, 5, 6) is $(1 \cdot 4) + (2 \cdot 5) + (3 \cdot 6)$ which equals 4 + 10 + 18 which gives a final result of 32.

  The other two overloaded multiplication operators allow multiplication of a `Vector3` with a floating point constant. For example, multiplying the vector (1, 2, 3) by 3 results in the vector (3, 6, 9). Two overloaded operators are needed here, one to handle multiplication of the vector by the constant in that order, the other to handle multiplication of the constant by the vector in that order. The results should be the same in both cases. As with other arithmetic operators, the operands should not be altered.

- `operator[]`

  The indexing operator should be overloaded to provide accessor methods for the class. Subscript 0 should provide access to the x component value of the vector; subscript 1 to the y component and subscript 2 to the z component. Subscript values other than 0, 1, or 2 produce undefined results. For speed, no error checking needs to be done.

  The choice of how the data members are implemented in the class could greatly affect the complexity of overloading this particular operator.

  Don't forget that this operator needs to be overloaded twice, once for getting a value and once for setting a value.

- `operator==`

  The equality operator should be overloaded to compare two `Vector3`s. The two vectors are considered equal only if they are componentwise equal. For example, (1, 2, 3) is equal to (1, 2, 3), but not to (4, 3, 2) or to (3, 2, 1). All components must be equal. The operands should not be altered.

Output

A driver program, `assign4.cpp`, is provided for this assignment. The purpose of a driver program is to test other pieces that you code. You do not need to write the driver program yourself. A copy of the driver program can also be found on `turing` at
`/home/turing/t90kjm1/CS241/Code/Fall2016/Assign4/assign4.cpp`.

```
/********************************************************************
   PROGRAM:    CSCI 241 Assignment 4
   PROGRAMMER: your name
   LOGON ID:   your z-ID
   DUE DATE:   due date of assignment

   FUNCTION:   This program tests the functionality of the Vector3
               class.
********************************************************************/

#include <iostream>
#include "Vector3.h"

using std::cout;
using std::endl;

int main()
   {
   int test = 1;

   cout << "\nTest " << test++ << ": Constructor and printing\n" << endl;

   const Vector3 v1, v2(1.0, 2.0, 3.0);

   cout << "v1: " << v1 << endl;
   cout << "v2: " << v2 << endl;

   cout << "\nTest " << test++ << ": Addition and subtraction\n" << endl;

   const Vector3 v3(1.0, 2.0, 3.0), v4(-2.0, 3.0, -1.0);

   cout << "v3: " << v3 << endl;
   cout << "v4: " << v4 << endl << endl;

   cout << "v3 + v4 is " << v3 + v4 << endl;
   cout << "v3 - v4 is " << v3 - v4 << endl;

   cout << "\nTest " << test++ << ": Vector multiplication\n" << endl;

   cout << "The scalar product of " << v3 << " and " << v4 << " is ";
   cout << v3 * v4 << endl;

   cout << "\nTest " << test++ << ": Scalar multiplication\n" << endl;

   float k = 2.345;

   cout << v3 << " * " << k << " = " << v3 * k << endl;
   cout << k << " * " << v4 << " = " << k * v4 << endl;

   cout << "\nTest " << test++ << ": Subscripting\n" << endl;

   const Vector3 v5(3.2, -5.4, 5.6);
   Vector3 v6(1.3, 2.4, -3.1);

   cout << "v5: " << v5 << endl;
   cout << "v6: " << v6 << endl;

   cout << "v5[0] = " << v5[0] << endl;
   cout << "v5[1] = " << v5[1] << endl;
   cout << "v5[2] = " << v5[2] << endl;
   v6[0] = -2.4;
   v6[1] = -1.3;
   v6[2] = 17.5;

   cout << "v6: " << v6 << endl;
   v6 = v5;
   cout << "v6: " << v6 << endl;

   cout << "\nTest " << test++ << ": Equality\n" << endl;

   const Vector3 v7(-1, 2, -1), v8(-1, 2, -2);

   cout << v7 << " and " << v7 << " are ";

   if (v7 == v7)
      cout << "equal" << endl;
   else
      cout << "not equal" << endl;

   cout << v7 << " and " << v8 << " are ";

   if (v7 == v8)
      cout << "equal" << endl;
   else
      cout << "not equal" << endl;

   return 0;
   }
```

Output from the correctly functioning driver program should look like the following:

```
Test 1: Constructor and printing

v1: (0, 0, 0)
v2: (1, 2, 3)

Test 2: Addition and subtraction

v3: (1, 2, 3)
v4: (-2, 3, -1)

v3 + v4 is (-1, 5, 2)
v3 - v4 is (3, -1, 4)

Test 3: Vector multiplication

The scalar product of (1, 2, 3) and (-2, 3, -1) is 1

Test 4: Scalar multiplication

(1, 2, 3) * 2.345 = (2.345, 4.69, 7.035)
2.345 * (-2, 3, -1) = (-4.69, 7.035, -2.345)

Test 5: Subscripting

v5: (3.2, -5.4, 5.6)
v6: (1.3, 2.4, -3.1)
v5[0] = 3.2
v5[1] = -5.4
```

```
v5[2] = 5.6
v6: (-2.4, -1.3, 17.5)
v6: (3.2, -5.4, 5.6)

Test 6: Equality

(-1, 2, -1) and (-1, 2, -1) are equal
(-1, 2, -1) and (-1, 2, -2) are not equal
```

Implementation Hints

The driver program should not be modified for your final submission. But while you're developing, modifying the driver program can definitely be in your best interest. The best way to modify the driver is to purposely comment out sections of code that you know you haven't implemented yet and that will only get in the way.

Start by commenting out everything in `main()`. Then move the top of the comment block past the initial `Vector3` variable declaration. This will test the constructor. Once the constructor compiles (no way to test it yet), move the start of the comment block down past the initial printing. Implement the overloaded stream insertion operator.

Continue this process, uncommenting small parts of the driver program and implementing what is needed. Little by little a fully functioning class and program is implemented.

Other Points

- Unlike Assignments 2 - 3, there is no data file for this assignment. You're free to develop your code wherever you like.
- A `Makefile` is required. Same as always. Make sure it has appropriate rules for all the pieces involved. It will be very similar to the one used for Assignment 3.
- The class should have a header file for its class definition and a source code file for its implementation. Note that the driver program assumes that your header file is called `Vector3.h`. If you name your header file differently, you'll need to modify the driver program accordingly.
- All overloaded operator functions of the class that can be implemented as methods should be implemented as methods.
- The name of your source code file should be `assign4.cpp`.
- As always, programs that do not compile on `turing/hopper` automatically receive 0 points.
- Make sure to document your program according to the standards listed in the Course Notes book. In particular, each function should have a documentation box describing the purpose of the function, the input parameters, and the return value (if any). There should also be a documentation box for the program as a whole.
- Submit your program using the electronic submission guidelines posted on the course web site and described in class.