Homework 1

PHY 250, Winter 2024

Due: 02-04-2024

Assignment

The goal of this assignment is to solidify the concepts we have discussed over the last few weeks by writing a *three vector* class. Specifically, inside of your namespace in your MyClass.hpp file create a new class called vec3 with the following properties,

1. Three floats called x, y, and z.
2. A default constructor that initializes all three values to 0.0.
3. An explicit constructor that takes three floats and sets x, y, and z appropriately.
4. A destructor. Because we are not doing any dynamic memory allocation, it is not strictly necessary to write a destructor; however, we will include one for good practices. It will just be,
   <your namespace>::vec3::~vec3(){}

Discussion

To make our lives easier we will begin by overloading the `[]` operator. For our vecN class we had a float* so it made sense to use

```
1 float& PHY::vecN::operator[](const int index){
2     return m_vals[index];
3 }
```

This will be slightly different for our vec3 class. Because the only variables that our vec3 class has are float x, y, z; they will be stored continuously in memory.

The `[]` operator works in the following way. Let's say we make an array of five floats,

```
1     float arr[5];
```

What we are doing is asking for a block of memory that can hold 5 floats. We may reserved the following block,

0xfe0	0xfe4	0xfe8	0xfec	0xff0

We treat arr as an array but reall it is just a float* that points to 0fe0. When we use the `[]` operator with a variable say

```
1     float x = arr[2];
```

What we are saying is,

1. go to address arr points to, 0xfe0.
2. move forward in memory two blocks, 0xfe8
3. give me the float at, dereference, this block of memory, *(0xfe8)

The following statements are equivalent

```
1     float x = arr[2];
2     float x = *(arr + 2);
```

The point to all of this was to try and explain why we can do the following for our vec3 class.
To overload the `[]` operator in our vec3 class use the following code in your .cpp file,

```
1 float& vec3::operator[](const int index){
2     return (&x)[index];
3 }
```

You will also need to include the appropriate definition in your .hpp file

Assignment cont.

Your vec3 class should also have the following member functions and operators.

4. Overload the `[]` operator to return a float&.
5. Overload the +, -, *, / operators to perform element wise addition, subtraction, multiplication, and division. They should take a const vec3& as an argument and return a vec3.
6. Overload the +=, -= operators to perform element wise =+, -=. They should take a const vec3& as an argument and return a vec3&.
7. Overload the *=, /= operators to scale all elements. They should take a const float as an argument and return a vec3&.
8. Overload the == to perform element wise comparison. It should take a const vec3& as an argument and return a bool.
9. Write a member function that computes the length of your vec3, l2 norm. It should return a float.
10. Write a member function that normalizes your vec3. It should return a vec3&.

We can also overload operators out side of the class, this is useful for operations where the order is ambiguous, e.g.

$$a * x == x * a$$

Outside of the scope of your vec3 class, but still inside your namespace, include the following lines in your .hpp file.

```
class vec3{
...
};

vec3 operator*(const float scale, const vec3& other);
vec3 operator*(const vec3& other, const float scale);
```

Then in your .cpp file, replacing PHY with your namespace

```
PHY::vec3 PHY::operator*(const float scale, const vec3& other){
    PHY::vec3 ret(other);
    ret *= scale;
    return ret;
}
PHY::vec3 PHY::operator*(const vec3& other,const float scale){
    PHY::vec3 ret(other);
    ret *= scale;
    return ret;
}
```

A nonmember function or operator refers to a function that acts on a class but is defined outside of class scope, for us still inside namespace.

11 Write a nonmember function that computes the dot product of two vec3.

12 Write a nonmember function that computes the cross product of two vec3.

13 Write a nonmember function that computes the distance, l2, between two vec3.

14 Overload the / operator for a vec3, just like the example but division.

15 Overload the >=, <= operators to element wise compare a vec3 with a float.

Lastly, write a new class called vec4. It will contain four floats x, y, z, w;
it should have

1. A default constructor that initializes all four values to 0.0.

2. An explicit constructor that takes four floats and sets x, y, z, and w appropriately.

3. A destructor. Again just PHY::vec4::∼vec4(){}

We will now go back to our old friend the = operator. Because we are not dynamically allocating any memory, there is no fear of a memory leak so you do not need to worry about safety.

16 For your vec4, inside the class scope, overload the = operator. It should take a const vec3& as an argument and return a vec4&.

GLHF