CS 538: Assignment 02

Programming Assignments (90%)

Setup

- **This assignment is in C++.**
- **Pull and merge from my repository to get the updated CMakeLists.txt file and the TestVec3.cpp file.**
- Add a new file to src/include called **Vector.hpp** (if you don't already have it from the exercises).
- Copy folder shaders/BasicRenderApp as **shaders/PotatoRenderApp**
- Copy src/include/BasicRenderEngine.hpp to **src/include/PotatoRenderEngine.hpp**
    - Change all references to BasicRenderEngine to PotatoRenderEngine
    - Add an include to "Vector.hpp"
- Copy src/lib/BasicRenderEngine.cpp to **src/lib/PotatoRenderEngine.cpp**
    - Change all references to BasicRenderEngine to PotatoRenderEngine
- Copy src/app/BasicRenderApp.cpp and name it **src/app/PotatoRenderApp.cpp**
    - Change all references to BasicRenderEngine to PotatoRenderEngine
    - At lines 197 and 198, change BasicRenderApp to PotatoRenderApp
- **Make sure that you follow the instructions for installing doctest in the updated instruction slide deck!**
- Make sure the sample **configures**, compiles, and runs as-is
- You should also be able to run **test programs** now in the testing side window.

Vector.hpp

For this assignment, you will focus on implementing the Vec3 struct such that it passes the tests provided. ***The implementation will be slightly different from the slides***, but the changes should be apparent from the descriptions below.

Make sure you at least have the following includes:

```
#pragma once
#include <cmath>
#include <string>
using namespace std;
```

All definitions will be under the **potato** namespace.

VecLength

Create the **VecLength** struct as we defined in the slides so that when T = double, **type** will be double. Otherwise, **type** will be float.

Vec3

Create Vec3 as a templated struct: **template<typename T> struct Vec3**

NOTE: If you are able to pass the tests making it a class instead of a struct, you may do so.

The struct should contain the following data:

- x, y, z values of type T
- All values should initialize to zero.
- You should create the values such that they are ***unioned*** with the following:
    - x = r = s
    - y = g = t
    - z = b = p

The struct should also contain the following member functions:

- **Vec3()**
    - Default constructor
- **Vec3(T x, T y, T z)**
    - Should initialize the values to given values
- **template<typename U>**
  Vec3(const Vec3<U> &other)
    - Should initialize to the values from the other Vec3.
    - Use static_cast<T> to convert values to appropriate type.
- **template<typename U>**
  auto operator+(const Vec3<U> &v2) const -> Vec3<decltype(T{} + U{})>
    - Overloads addition operator; add two Vec3's and return answer
- **template<typename U>**
  auto operator-(const Vec3<U> &v2) const -> Vec3<decltype(T{} - U{})>
    - Overrides subtraction operator: subtract two Vec3's and return answer
- **template<typename U>**
  auto operator*(const Vec3<U> &v2) const -> Vec3<decltype(T{} * U{})>
    - Overloads multiplication operator: multiply components PIECEWISE and return answer
- **template<typename U>**
  auto operator*(const U &s) const -> Vec3<decltype(T{} * U{})>
    - Overloads multiplication by scalar operator: multiply each component by scalar and return answer

- **template<typename U>**
  auto operator/(const U &s) const -> Vec3<decltype(T{} / U{})>
  - Overloads division by scalar operator: divide each component by scalar and return answer
- **T operator[](int i) const**
  - If the index is within bounds, return the appropriate component.
  - Otherwise, throw an out_of_range exception
- **T &operator[](int i)**
  - If the index is within bounds, return the appropriate component.
  - Otherwise, throw an out_of_range exception
- **friend ostream& operator<<(ostream& os, const Vec3<T> &v)**
  - Overloads << operator (used in things like cout)
  - "Print" to os: "(" << x << "," << y << "," << z << "," << w << ")"
  - Do NOT use an endl!
  - Return os
- **template<typename U>**
  auto dot(const Vec3<U> &v2) const -> decltype(T{} * U{})
  - Compute and return the dot product
- **template<typename U>**
  auto cross(const Vec3<U> &v2) const -> Vec3<decltype(T{} * U{})>
  - Compute and return the cross product
- **auto length() const -> typename VecLength<T>::type**
  - Compute and return the length of the Vec3
- **auto normalize() const -> Vec3<typename VecLength<T>::type>**
  - Create and return the normalized version of the current vector.
  - **Do NOT modify the existing vector values! This is meant to return a NEW Vec3!**

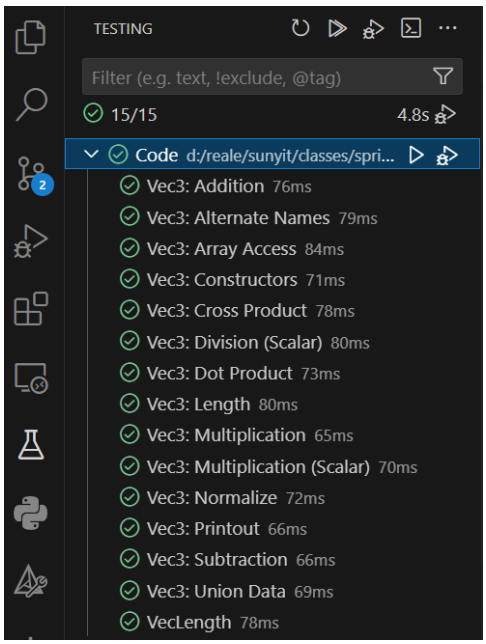
Aliases for Vec3 Variants

Also add the following variants for Vec3:

- **using Vec3f = Vec3<float>;**
- **using Vec3i = Vec3<int>;**
- **using Vec3d = Vec3<double>;**
- **using Vec3u = Vec3<unsigned char>;**

Testing Screenshot (10%)

- **Take a screenshot** of the result of running the tests.
- Save this into the **screenshots** folder as "Assign02.png"



Grading

Your OVERALL assignment grade is weighted as follows:

- 90% - Programming
- 10% - Screenshot