ECE 4122/6122 **Advanced Programming Techniques**

Short Exercise #1 Creating a Custom Type in C++

Assigned: Wednesday, September 5, 2018 *Due: Friday, September 14, 2018*

In this exercise you will be creating a custom 3-dimensional vector type made of 32 bit floats. Later we expand on its capabilities with templates. You will not be provided with any skeleton code however you **MUST** follow the given guidelines on folder structure and file naming in order to make it easier to grade. Points will be deducted if you do not do this.

The main folder of your solution should be named se1 (note capitalization) and should follow the follwing structure :

```
se1
├── src
│   ├── vector3.h
│   └── vector3.cc
├── CMakeList.txt
└── main.cc
```

vector3.h and *vector3.cc* will be where you implement your vector class. *main.cc* is where you will write code to test your vector class and *CMakeList.txt* will be used to help build your solution. This assignment is short enough where we won't be submitting jobs to the cluster but will just be developing and debugging on the login node so there is no .pbs file.

You will be making your custom vector3 type as a *struct*. Note that c++ supports classes and structs and the only difference between the two, as far as the language is concerned, is that by default the members of a struct are public while the members of a class are private by default. Otherwise they are equivalent. Since we're going to make the coordinates (x, y, z) of our vector public, we'll use a struct.

You should start your vector3.h file like this :

```
struct Vector3 {
    float x;
    float y;
    float z;

    //methods go here
};
```

You now need to implement the following methods for your custom type :

```
Vector3() = default                          //default constructor
Vector3(float xyz);                          //set x, y, and z to xyz
Vector3(float x, float y, float z)           //set component by name

Vector3 operator+(const Vector3& rhs);       //component-wise add
Vector3 operator-(const Vector3& rhs);       //component-wise subtract
Vector3 operator*(const Vector3& rhs);       //component-wise multiplication
Vector3 operator/(const Vector3& rhs);       //component-wise division

Vector3 operator+(float rhs);                //add rhs to each component
Vector3 operator-(float rhs);                //subtract rhs from each component
Vector3 operator*(float rhs);                //multiply each component by rhs
Vector3 operator/(float rhs);                //divide each component by rhs

float operator|(const Vector3& rhs);         // dot product
Vector3 operator^(const Vector3& rhs);       // cross product

Vector3& operator+=(const Vector3& rhs);     //component-wise add
Vector3& operator-=(const Vector3& rhs);     //component-wise subtract
Vector3& operator*=(const Vector3& rhs);     //component-wise multiplication
Vector3& operator/=(const Vector3& rhs);     //component-wise division

//  Vector3++ and ++Vector3 rotate xyz to the right
//  i.e. make x = z, y = x, z = y
//  Make sure they function correctly ++v vs v++
Vector3& operator++();
```

```
Vector3 operator++(int __unused);

//  Vector3-- and --Vector3 rotate xyz to the left
//  i.e. make x = y, y = z, z = x
Vector3& operator--();
Vector3 operator--(int __unused);

bool operator==(const Vector3& rhs);        //component-wise equality
bool operator!=(const Vector3& rhs);        //component-wise inequality
```

Your *CMakeList.txt* file should contain the following :

```
cmake_minimum_required(VERSION 3.12)
project(se1)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Werror")

include_directories(
    src
)

set(SOURCE_FILES
    src/vector3.h
    src/vector3.cc
)

add_executable(${PROJECT_NAME} ${SOURCE_FILES} main.cc)
```

Your *main.cc* file should contain code you use to test your implementation to verify it is working correctly. We will be using another *main.cc* when we test your solutions.

To build your solution, make sure you load the following modules on pace-ice

```
module load gcc/7.3.0
module load cmake/3.2.0
```

To test your implementation, create a build folder in your project directory :

```
se1
├── build
├── src
│   ├── vector3.h
│   └── vector3.cc
├── CMakeList.txt
└── main.cc
```

From within your build folder, type **cmake ..** . Assuming this command completes successfully, your build folder will contain a Makefile. Type **make** to build your project. All projects must compile without any warnings or errors to receive full credit. To execute your program type **./se1** from within the build directory.

For submission, compress your solution into se1.zip (NOT .TAR.GZ, or .RAR or anything else). zip is installed on the login node. Submit via canvas.