Implementing The Geometry Library

You need to implement the basic methods for a collection of classes, which represent geometric objects we will use a lot in this course. These basic methods are easy to implement, but a bit cumbersome to use, so there are other more convenient methods that call the basic methods.

The methods are declared in the following header files:

- `Float4.h`
- `Point4.h`
- `Vector4.h`

The classes are implemented in the following files:

- `Float4.cpp`
- `Point4.cpp`
- `Vector4.cpp`

Some methods are NOT implemented. You must implement them. All those methods should be implemented in the file `GeomLib.cpp` .

Two test files are provided:

1. `viewMatrix.cpp` : defines the view matrix. The output of running this program is here: `viewmatrix-out.txt` .
2. `test.cpp` : runs some simple tests. You should probably expand this code.

The Classes

- `Float4` : Contains an array of 4 floats. Methods are provided to access the four components X, Y, Z, and W, as well as to compare two `Float4`s and for stream I/O.

- `Point4` : This is a subclass of `Float4`, and represents a point in space (a 3D position). The W component of the array is guaranteed to be 1.0f. Some +/- methods are provided (only those that are valid for points and vectors).

- `Vector4` : This is a subclass of `Float4`, and represents a vector in space (a 3D displacement). The W component of the array is guaranteed to be 0.0f. Some +/-/* methods are provided (again, only those that are valid for points and vectors).

- `Matrix4` : Contains an array of 4 `Float4`s, for a total of 16=4x4 floats. It represents a 4x4 homogeneous transformation matrix.

Convenience Methods

The following convenience methods are implemented for you. They will use the methods that YOU must implement.

- operator[]: lets you access components of a `Float4`. example:

```
Point4 p(1,2,3);
Vector4 v(1,2,3);
Matrix4 m;
p[0] = p[1] + v[2]; // ie, px = py + vz
m[0][0] = 1;        // change top left entry
m[3][3] = 2;        // change bottom right entry
```

- named accessor methods. example:

```
Point4 p(1,2,3);
Vector4 v(1,2,3);
p.X() = p.Y() + v.Z() * p.W(); // ie, px = py + vz*1
```

- Vector/point/scalar arithmetic: examples:

```
Point4 p(1,2,3);
Vector4 v(1,2,3);
p = p + v;
p = p - v;
p = p + 2*v;
v = v/5;
v = -v;
v = v.normalized();  // same as v = v / v.length()
p = -p;   //              gives a compile-time error
p = p*2;  // nonsensical, gives a compile-time error
p = v/2;  // nonsensical, gives a compile-time error
p = p + p;// nonsensical, gives a compile-time error
```

- Dot and cross product: example:

```
Vector4 v1(1,2,3);
Vector4 v2(2,1,0);
Vector4 v3 = v1 ^ v2; // cross product
float   d  = v1 * v2; // dot product
```

- Matrix methods:

```
Matrix4 m = Matrix4::XRotation(45);
Float4 p1(1,2,3,4);
Float4 p2 = m * p1; // multiplies matrix x p1
Point4 p  = p1;     // run-time error, p2.W() != 1
Point4 q  = p1.homogenized(); // no error
Matrix4 m2 = Matrix4(1,0,0,0, 0,1,0,1, 1,2,3,4, 0,0,0,1);
Matrix4 m3 = m * m2; // matrix x matrix
```

Methods YOU must implement

The following methods should be implemented in `GeomLib.cpp` :

```
// Copy all components of "other" into this point.
void Float4::copyFrom(const Float4& other)

// Add all four components of this and "other" into sum.
void Float4::plus(const Float4& other, Float4& sum) const

// Subtract all four components of "other" from this point, into difference
void Float4::minus(const Float4& other, Float4& difference) const

// Dot product of all four (X Y Z W) components of "other" and this point.
float Float4::dot(const Float4& other) const

// Divide all 4 of this point components by W (if W==0, do nothing),
// into result.
void Float4::homogenize(Float4& result) const

// Multiply all four (X Y Z W) components of this point by scale factor,
// into result.
void Float4::times(float factor, Float4& result) const

// return the distance from this point to "other".
float Point4::distanceTo(Point4& other) const

// Length, using three (X Y Z) components of this point.
float Vector4::length() const

// Divide first three (X Y Z) components of this point by length,
// into result (if length==0, do nothing)
void Vector4::normalize(Vector4& result) const

// Cross product of this vector x "other", into result
// Use only X Y Z components.
// Sets W component of product to 0.
void Vector4::cross(const Vector4& other, Vector4& result) const

// return the angle (in radians) between this vector and "other".
// If this or other is (0 0 0), return 0 angle.
float Vector4::angle(Vector4& other) const

// Copy all components of other matrix into this.
void Matrix4::copyFrom(const Matrix4& other)

// Add all components of this and "other" matrix into sum
void Matrix4::plus(const Matrix4& other, Matrix4& sum) const
```

```
// Subtract all components of this matrix minus "other", into difference
void Matrix4::minus(const Matrix4& other, Matrix4& difference) const

// Multiply all components by factor, into product
void Matrix4::times(float factor, Matrix4& product) const

// Multiply (*this) x ("other"), and put product into product
// CAREFULL!! this may == &product!  So watch for partially-modified entries.
// Store result in temp array first!
void Matrix4::times(const Matrix4& other, Matrix4 &product) const

// Multiply (*this) x ("point"), and put resulting point into "product".
// CAREFULL!! point may == product, so watch for partially-modified entries.
// Store result in temp array first!
void Matrix4::times(const Float4& point, Float4& product) const

// Transpose this matrix, put result into result.
//
// CAREFULL!! this may == &result!  So watch for partially-modified entries.
// Store result in temp array first!
void Matrix4::transpose(Matrix4& result) const

// Set this matrix to identity matrix.
void Matrix4::setToIdentity()

// Set this matrix to rotation about X axis.
// "angle" is in DEGREES.
void Matrix4::setToXRotation(float angle)

// Set this matrix to rotation about Y axis.
// "angle" is in DEGREES.
void Matrix4::setToYRotation(float angle)

// Set this matrix to rotation about Z axis.
// "angle" is in DEGREES.
void Matrix4::setToZRotation(float angle)

// Set this matrix to translation matrix.
void Matrix4::setToTranslation(float tx, float ty, float tz)

// Set this matrix to scaling matrix.
void Matrix4::setToScaling(float sx, float sy, float sz)
```