Programming Assignment #1

CS 200, Fall 2017

Due Wednesday, September 13

I will provide you with a header file named `Affine.h`, that contains the following declarations and definitions.

```
struct Hcoord {
  float x, y, w;
  explicit Hcoord(float X=0, float Y=0, float W=0);
  float& operator[](int i) { return *(&x+i); }
  float operator[](int i) const { return *(&x+i); }
  static bool near(float x, float y) { return std::abs(x-y) < 1e-5f; }
};

struct Point : Hcoord {
  explicit Point(float X=0, float Y=0);
  Point(const Hcoord& v) : Hcoord(v) { assert(near(w,1)); }
};

struct Vector : Hcoord {
  explicit Vector(float X=0, float Y=0);
  Vector(const Hcoord& v) : Hcoord(v) { assert(near(w,0)); }
};

struct Affine {
  Hcoord row[3];
  Affine(void);
  Affine(const Vector& Lx, const Vector& Ly, const Point& disp);
  Hcoord& operator[](int i) { return row[i]; }
  const Hcoord& operator[](int i) const { return row[i]; }
};
```

```
Hcoord operator+(const Hcoord& u, const Hcoord& v);
Hcoord operator-(const Hcoord& u, const Hcoord& v);
Hcoord operator-(const Hcoord& v);
Hcoord operator*(float r, const Hcoord& v);
Hcoord operator*(const Affine& A, const Hcoord& v);
Affine operator*(const Affine& A, const Affine& B);
float dot(const Vector& u, const Vector& v);
float abs(const Vector& v);
Vector normalize(const Vector& v);
Affine rotate(float t);
Affine translate(const Vector& v);
Affine scale(float r);
Affine scale(float rx, float ry);
```

Note that the `Hcoord` structure represents homogeneous coordinates in general (where the w coordinate can take any value). The `Point`, and `Vector` structures, which use homogeneous coordinates, are derived from `Hcoord`. However, a `Point` must always have $w = 1$, and a `Vector` must always have $w = 0$. The functions given in this header file are described as follows.

`Hcoord::Hcoord(X,Y,W)` — (constructor) creates the three–component (homogeneous coordinate) vector $[X, Y, W]$.

`Hcoord::operator[](i)` — subscripting operator. Returns the i–th component of a homogeneous coordinate vector. If $i \neq 0, 1, 2$, the result is undefined. [Implemented.]

`Hcoord::near(x,y)` – convenience function to compare two floating point numbers: returns `true` if x and y are close enough to be considered equal. [Implemented.]

`Point::Point(X,Y)` — constructor to initialize the components of a point. Creates a point with components (X, Y).

`Point::Point(v)` – conversion operator to *attempt* to convert to a point. This will fail, and the program will crash, if $v_w \neq 1$. [Implemented.]

`Vector::Vector(X,Y)` — constructor to initialize the components of a vector. Creates a vector with components $\langle X, Y \rangle$.

`Vector::Vector(v)` – conversion operator to *attempt* to convert to a vector. This will fail, and the program will crash, if $v_w \neq 0$. [Implemented.]

`Affine::Affine` — default constructor. Creates the affine transformation corresponding to the trivial affine transformation whose linear part is the 0 transformation, and whose translation part is the 0 vector. Note that the resulting matrix is <u>not</u> the 3×3 matrix whose entries are all zeros; rather it is the same as the matrix for uniform scaling by 0 with respect to the origin, H_0.

`Affine::Affine(Lx,Ly,D)` — constructor to initialize an affine transformation. The quantities `Lx`, `Ly`, `D` give the values of the *columns* of the transformation.

`Affine::operator[](i)` — subscripting operator. Returns the i–th row of an affine transformation. [Implemented.]

`operator+(u,v)` — returns the sum $\mathbf{u}+\mathbf{v}$ of two three–component vectors.

`operator-(u,v)` — returns the difference $\mathbf{u}-\mathbf{v}$ of two three–component vectors.

`operator-(v)` — returns the component–wise negation $-\mathbf{v}$ of a three–component vector.

`operator*(r,v)` — returns the product $r\mathbf{v}$ of a scalar and a three–component vector.

`operator*(A,v)` — returns the result $A\mathbf{v}$ of applying the affine transformation A to the three–component vector $\mathbf{v}$.

`operator*(A,B)` — returns the composition $A \circ B$ (matrix product) of the affine transformations A and B. Note that I'm looking for simplicity of coding here.

`dot(u,v)` — returns the dot product $\vec{u} \cdot \vec{v}$ of two–dimensional vectors.

`abs(v)` — returns the length $|\vec{v}|$ of a two–dimensional vector.

`normalize(v)` — returns the normalization of the two–dimensional vector $\vec{v}$; i.e., the unit vector $\vec{u}$ that points in the same direction as $\vec{v}$. The vector $\vec{v}$ is assumed to be non–zero. No error checking is performed.

`rotate(t)` — returns the affine transformation R_t for rotation by the angle t (in radians) with respect to the origin.

`translate(v)` — returns the affine transformation $T_{\vec{v}}$ for translation by the vector $\vec{v}$.

`scale(r)` — returns the affine transformation H_r for uniform scaling by r with respect to the origin.

`scale(rx,ry)` — returns the affine transformation $H_{\langle r_x, r_y \rangle}$ for inhomogeneous scaling by factors r_x and r_y with respect to the origin.

You are to implemented the functions in the above header file (except for the ones already implemented). Your implementation file should be named `Affine.cpp`. Only `Affine.h` and the standard header file `cmath` may be included (note that `Affine.h` already includes `cmath`); you may not alter the contents of this header file in any way.