1. Write the code to compute a matrix to rotate an image counterclockwise around its center by 1.1 radians.

We need this for source to dest: \( T(\frac{w}{2}, \frac{h}{2}) R(1.1) T(\frac{-w}{2}, \frac{-h}{2}) \)

Inverse: \( T(\frac{-w}{2}, \frac{-h}{2})^{-1} R(1.1)^{-1} \) \( T(\frac{w}{2}, \frac{h}{2})^{-1} = T(\frac{w}{2}, \frac{h}{2}) R(-1.1) T(-\frac{w}{2}, -\frac{h}{2}) \)

```cpp
CHMatrix m;
m.SetIdentity();

CHMatrix t1, r, t2;
t1.SetTranslate(w/2, h/2);
r.SetRotate(-1.1);
t2.SetTranslate(-w/2, -h/2);

m = t1 * r * t2;
```

2. Write the code to compute a matrix that will put the lower left corner of an image at \((100, 100)\) and the upper right corner at \((350, 200)\).

We'll scale to the right size. Then we'll rotate the diagonal onto the horizontal. Next we rotate the diagonal onto the right angle. Finally, we move the corner into place. Let's first build the source to destination matrices:

How big is it diagonally, now? From that we can compute the scale:

```cpp
double ll[2] = {100, 100};

// How big is it now?
double nowSize = sqrt(double(w * w + h * h));

// How big do we want it to be?
double wantSize = sqrt(pow(ur[0]-ll[0], 2) + pow(ur[1]-ll[1], 2));

double scale = wantSize / nowSize;
```

First, let's rotate the diagonal to the horizontal.

```cpp
    // Cos and sin to rotate diagonal to the horizontal
    double cos1 = w / nowSize;
    double sin1 = -h / nowSize;
```

Next, we need the sin and cos to rotate the horizontal to the destination angle:


// Cos and sin to rotate diagonal to the required angle
double cos2 = (ur[0] - ll[0]) / wantSize;
double sin2 = (ur[1] - ll[1]) / wantSize;

Finally, we translate it to (100, 100).

Writing these as matrices:

T(100, 100) R(cos2, sin2) R(cos1, sin1) S(scale)

Inverse:

S(scale)^-1 R(cos1, sin1)^-1 R(cos2, sin2)^-1 T(100, 100)^-1

= S(1/scale) R(cos1, -sin1) R(cos2, -sin2) T(-100, -100)

The whole thing:

double ll[2] = {100, 100};

// How big is it now?
double nowSize = sqrt(double(w * w + h * h));

// How big do we want it to be?
double wantSize = sqrt(pow(ur[0]-ll[0], 2) + pow(ur[1]-ll[1], 2));

double scale = wantSize / nowSize;

// Cos and sin to rotate diagonal to the horizontal
double cos1 = w / nowSize;
double sin1 = -h / nowSize;

// Cos and sin to rotate diagonal to the required angle
double cos2 = (ur[0] - ll[0]) / wantSize;
double sin2 = (ur[1] - ll[1]) / wantSize;

CHMatrix s, r1, r2, t;
s.setScale(1/scale, 1/scale);
  r1[0][0] = cos1;
  r1[1][1] = cos1;
  r1[0][1] = sin1;
  r1[1][0] = -sin1;

  r2[0][0] = cos2;
  r2[1][1] = cos2;
  r2[0][1] = sin2;
  r2[1][0] = -sin2;

t.setTranslate(-ll[0], -ll[1]);

CHMatrix m;

m = s * r1 * r2 * t;
3. Write the code for a mapping that will change a w by h image into a triangle, by having the top line scale down to a point on the top-left of the image.

```cpp
double y1 = y2;
double x1 = x2 * (1 - double(y2)/h);
```

4. Write the code for a mapping that will change a w by h image into a triangle, by having the top line scale down to a point in the center-top of the image.

```cpp
double y1 = y2;
double x1 = (x2 - w/2) * (1 - double(y2)/h) + w/2;
```

5. Write the mapping code for a swirl filter. This is a filter where the rotation of the image increases as you get farther from the center of the image.

```cpp
for(int y2=0; y2<h; y2++)
{
    for(int x2=0; x2<w; x2++)
    {
        double swirlFactor = 0.01;
        double d = sqrt(pow(x2-w/2, 2.0) + pow(y2-h/2, 2.0));

        CHMatrix r, t1, t2, m;
        t1.SetTranslate(-w/2, -h/2);
        r.SetRotate(d * swirlFactor);
        t2.SetTranslate(w/2, h/2);
        m = t2 * r * t1;
        // This is the equivalent point in the source image
        double x1 = m[0][0] * x2 + m[0][1] * y2 + m[0][2];
        double y1 = m[1][0] * x2 + m[1][1] * y2 + m[1][2];

        int ix = int(x1);
        int iy = int(y1);

        if(ix >= 0 && ix < w && iy >= 0 && iy < h)
        {
            m_image2[y2][x2*3] = m_image1[iy][ix*3];
            m_image2[y2][x2*3+1] = m_image1[iy][ix*3+1];
            m_image2[y2][x2*3+2] = m_image1[iy][ix*3+2];
        }
        else
        {
            // Outside area of source image
            m_image2[y2][x2*3] = 0;
            m_image2[y2][x2*3+1] = 0;
            m_image2[y2][x2*3+2] = 0;
        }
    }
}
```