// Illustrate the use of static member variables and static member functions
// Georgia Tech, ECE2036, Fall 2012

// We will define a class Point3 that manages a point in 3-D space, maintaining
// an x, y, and z coordinate. It also will maintain the "pixel index"
// that will index a 1-dimensional array of pixels that represents the
// set of points for an image.

// To do this, the conversion from x/y/z to pixel index requires
// knowledge of the width and height of the image. Since the width and
// height are the same for all instances of Point3 objects, it does not
// make sense to use normal member variables for these. What we want are
// variables that are part of the Point3 class, but are common for all
// instances. This is called a "static member variable", and is defined just
// like any other member variable, but is preceded by the "static"
// keyword. One other difference is that the static variables
// must later be "created", as illustrated in the Point3 class below.

// Since width and height are not in fact member variables for any
// specific Point3 instance, it would make sense that we have a way
// to set new values for these without requiring an existing object
// of class Point3. Recall that for normal member variables
// we must have a Point3 object to access a member variable or
// call a member function, such as:

// Point3 p(1, 2, 3);
// int x = p.GetX();
// int y = p.GetY();

// The above snippet calls member functions GetX and GetY for object
// "p" returning the x and y values for that specific Point3 object.

// To create functions that can be called WITHOUT an object, we create
// "static member functions". As in the static member variables, we
// simply precede the member function declaration with the "static"
// keyword. We can then call "SetW" (if SetW is static) without
// any existing objects of class Point3, as follows:

// Point3::SetW(256);
// Point3::SetH(256);

// or, if the W and H variables are public, we can just access them
// directly:

// Point3::W = 256;
// Point3::H = 256;

#include <iostream>
using namespace std;

// The following Point3 declaration would normally be put in Point3.h
// but is here for simplicity:

class Point3
{
 public:

 Program static-members.cc
Point3(int x0, int y0, int z0);
Point3(int x0, int y0); // Assumes z = 0;
Point3(int i0); // Initialize with index
// Accessor functions
public:
intGetX() const;
intGetY() const;
intGetZ() const;
intGetInd() const;
private:
// x, y, z, and ind are member variables, but private.
intx;
inty;
intz;
intind; // Pixel index value
// Now define the "static" width and height
private:
static intW; // Width of image
static intH; // Height of image
// Create the static "setter" functions for W and H
public:
static void SetW(int w0);
static void SetH(int h0);
};

// The following would be in Point3.cc
//
// Since we have static member variables W and H, we just actually
// define those and (optionally) initialize.
//
int Point3::W = 0;
int Point3::H = 0;

// Point3 constructors
Point3::Point3(int x0, int y0, int z0)
  : x(x0), y(y0), z(z0)
{  // Compute the pixel index
  ind = z * W * H + y * W + x;
}
Point3::Point3(int x0, int y0)
  : x(x0), y(y0), z(0)
{  // Compute the pixel index
  ind = z * W * H + y * W + x;
}
Point3::Point3(int i)
  : ind(i)
{  // i is the pixel index compute x, y, and z
  z = ind / (W * H);
  y = (ind - z * W * H) / W;
  x = ind % W;
}

Program static-members.cc (continued)
// Define the accessors
int Point3::GetX() const
{
    return x;
}

int Point3::GetY() const
{
    return y;
}

int Point3::GetZ() const
{
    return z;
}

int Point3::GetInd() const
{
    return ind;
}

// Define the static "setters" for W and H
void Point3::SetW(int w0)
{
    W = w0;
}

void Point3::SetH(int h0)
{
    H = h0;
}

// Main program for testing
int main()
{
    int imageW = 128;
    int imageH = 128; // Arbitrary values for testing
    Point3::SetW(imageW);
    Point3::SetH(imageH);
    int maxZ = 36; // Arbitrary value for testing
    int errorCount = 0;
    for (int i = 0; i < imageW * imageH * maxZ; ++i)
    {
        // Construct a point3 with all possible "ind" values
        Point3 pInd(i);
        // Construct another with the corresponding x, y, and z values
        Point3 pXYZ(pInd.GetX(), pInd.GetY(), pInd.GetZ());
        if (pXYZ.GetInd() != pInd.GetInd())
        {
            // OOps, mismath
            cout << "Error on index " << i << " XYZ.GetInd " << pXYZ.GetInd() << " Ind.GetInd " << pInd.GetInd() << " x " << pXYZ.GetX() << " y " << pXYZ.GetY() << " z " << pXYZ.GetZ() << endl;
        }
    }
    return 0;
}
if (errorCount != 0) {
    cout << "Found " << errorCount << " errors in index test" << endl;
}
errorCount = 0;
// Test the other way, starting with XYZ and converting to ind
for (int z0 = 0; z0 < maxZ; z0++)
{
    for (int y0 = 0; y0 < imageH; ++y0)
    {
        for (int x0 = 0; x0 < imageW; ++x0)
        {
            Point3 pXYZ(x0, y0, z0);
            Point3 pInd(pXYZ.GetInd());
            // Now see if the x, y, and z match
            if (pXYZ.GetX() != pInd.GetX() ||
                pXYZ.GetY() != pInd.GetY() ||
                pXYZ.GetZ() != pInd.GetZ())
            {
                cout << "Error on index " << pInd.GetInd() << ", x1 " << pXYZ.GetX() << ", y1 " << pXYZ.GetY() << ", z1 " << pXYZ.GetZ() << ", x2 " << pInd.GetX() << ", y2 " << pInd.GetY() << ", z2 " << pInd.GetZ() << endl;
                errorCount++;
            }
        }
    }
    if (errorCount != 0) {
        cout << "Found " << errorCount << " errors in xyz test" << endl;
    }
}