1 // Illustrate the use of static member variables and static member functions
2 // George F. Riley, Georgia Tech, Spring 2012
3
4 // We will define a class Point3 that manages a point in 3-D space, maintaining
5 // an x, y, and z coordinate. It also will maintain the "pixel index"
6 // that will index a 1-dimensional array of pixels that represents the
7 // set of points for an image.
8 //
9 // To do this, the conversion from x/y/z to pixel index requires
10 // knowledge of the width and height of the image. Since the width and
11 // height are the same for all instances of Point3 objects, it does not
12 // make sense to use normal member variables for these. What we want are
13 // variables that are part of the Point3 class, but are common for all
14 // instances. This is called a "static member variable", and is defined just
15 // like any other member variable, but is preceded by the "static"
16 // keyword. One other difference is that the static variables
17 // must later be "created", as illustrated in the Point3 class below.
18 //
19 // Since width and height are not in fact member variables for any
20 // specific Point3 instance, it would make sense that we have a way
21 // to set new values for these without requiring an existing object
22 // of class Point3. Recall that for normal member variables
23 // we must have a Point3 object to access a member variable or
24 // call a member function, such as:
25 //
26 // Point3 p(1, 2, 3);
27 // int x = p.GetX();
28 // int y = p.GetY();
29 //
30 // The above snippet calls member functions GetX and GetY for object
31 // "p" returning the x and y values for that specific Point3 object.
32 //
33 // To create functions that can be called WITHOUT an object, we create
34 // "static member functions". As in the static member variables, we
35 // simply preceed the member function declaration with the "static"
36 // keyword. We can then call "SetW" (if SetW is static) without
37 // any existing objects of class Point3, as follows:
38 //
39 // Point3::SetW(256);
40 // Point3::SetH(256);
41 //
42 // or, if the W and H variables are public, we can just access them
43 // directly:
44 //
45 // Point3::W = 256;
46 // Point3::H = 256;
47
48 #include <iostream>
49 using namespace std;
50
51 // The following Point3 declaration would normally be put in Point3.h
52 // but is here for simplicity:
53
54 class Point3
55 {
56 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:
int GetX() const;
int GetY() const;
int GetZ() const;
int GetInd() const;
private:
// x, y, z, and ind are member variables, but private.
int x;
int y;
int z;
int ind;       // Pixel index value
// Now define the "static" width and height
private:
static int W; // Width of image
static int H; // 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;
}
// 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(pIndGetX(), pIndGetY(), pIndGetZ());
        if (pXYZ.GetInd() != pInd.GetInd())
        {
            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;
}
```cpp
    errorCount++;
  }
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 (pXYZGetX() != pIndGetX() ||
          pXYZGetY() != pIndGetY() ||
          pXYZGetZ() != pIndGetZ())
        {
          cout << "Error on index " << pInd.GetInd()
               " x1 " << pXYZGetX()
               " y1 " << pXYZGetY()
               " z1 " << pXYZGetZ()
               " x2 " << pIndGetX()
               " y2 " << pIndGetY()
               " z2 " << pIndGetZ()
               " endl;
          errorCount++;
        }
    }
  }
if (errorCount != 0)
  {
    cout << "Found " << errorCount << " errors in xyz test" << endl;
  }
```