```
// ImageScaling.cpp - PEEPHOLE SYSTEM IMAGE SCALING FUNCTION
//
// MODULE INDEX
// NAME                 CONTENTS
// readImage            Read image data from file
// WriteImage           Write image data to file
// CoorToOfs            Convert coordinate to image data array offset
// ImageScaling         Scale a peephole image
// main                 Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 05-10-17     SHT     Original
// 25-10-17     SHT     Corrected integer overflow in 32-bit system
// 26-10-17     SHT     Corrected arithmetic exception on negative source offset
//
//-----------------------------------------------------------------------------

#include <cstdio>           // C-style standard input/output functions
#include <cstdlib>          // C-style standard library
#include <cstring>          // C-style string manipulation functions
#include <iostream>         // C++ input/output stream declarations
using namespace std;        // Expand the standard namespace

//-----------------------------------------------------------------------------

#define HDR_SIZ     12      // Peephole image header size
                            // 4 - PH24 magic header
                            // 4 - width
                            // 4 - height

#define PIX_SIZ     3       // Peeohole image pixel size (RGB component)

//-----------------------------------------------------------------------------

// READ IMAGE DATA FROM FILE

unsigned char*
ReadImage (
    const char *fnm)        // Filename
{
    long size;
    unsigned char *tmp;

    FILE *f = fopen(fnm, "rb");
    fseek(f, 0, SEEK_END);
    size = ftell(f);
    fseek (f, 0, SEEK_SET);

    tmp = (unsigned char*)malloc(size);

    fread(tmp, 1, size, f);
    fclose(f);
```

```
    return tmp;
}

//---------------------------------------------------------------------------

// WRITE IMAGE DATA TO FILE

void
WriteImage (
    const char              *fnm,   // Filename
    const unsigned char     *img,   // Image data buffer
    const int               len)    // Buffer length
{
    FILE *f = fopen(fnm, "wb");
    fwrite(img, 1, len, f);
    fclose(f);
}

//---------------------------------------------------------------------------

// CONVERT COORDINATE TO IMAGE DATA ARRAY OFFSET

int
CoorToOfs(
    int x,                          // X coordinate
    int y,                          // Y coordinate
    int width)                      // Image width
{
    return HDR_SIZ + (y * width * PIX_SIZ) + (x * PIX_SIZ);
}

//---------------------------------------------------------------------------

// SCALE A PEEPHOLE IMAGE

unsigned char*
ScaleImage(
    const unsigned char *unscaledImage,     // Pointer to source image data
    long                xOfs,               // X offset of source image
    long                yOfs,               // Y offset of source image
    unsigned long       unscaledWidth,      // Width of rect to crop
    unsigned long       unscaledHeight,     // Height of rect to crop
    unsigned long       scaledWidth,        // Destination image width
    unsigned long       scaledHeight,       // Destination image height
    unsigned char       fillRed,            // Red component of fill color
    unsigned char       fillGreen,          // Green component of fill color
    unsigned char       fillBlue)           // Blue component of fill color
{
    unsigned long       srcWidth;           // Source image width
    unsigned long       srcHeight;          // Source image height
    long                fX, fY;             // From coordinate
    long                tX, tY;             // To coordinate
    long                srcX, srcY;         // Source coordinate
    unsigned long       dstX, dstY;         // Destination coordinate
```

```
unsigned long       wX, wY;             // Weight in x and y direction
unsigned long       srcOfs, dstOfs;     // Data offset
unsigned long long  w;                  // Weight
unsigned long long  totW;               // Total weights
unsigned long long  totR;               // Total red * weight product
unsigned long long  totG;               // Total green * weight product
unsigned long long  totB;               // Total blue * weight product
unsigned char       red;                // Red color value
unsigned char       green;              // Green color value
unsigned char       blue;               // Blue color value
unsigned char       *dst;               // Destination image data buffer

// Verify source image header

if (
    unscaledImage[0] != 'P' ||
    unscaledImage[1] != 'H' ||
    unscaledImage[2] != '2' ||
    unscaledImage[3] != '4'
) {
    cerr << "Invalid file header" << endl;
    exit(1);
}

// Retrieve source image dimension

srcWidth = unscaledImage[4] << 24
         | unscaledImage[5] << 16
         | unscaledImage[6] << 8
         | unscaledImage[7];

srcHeight = unscaledImage[8] << 24
          | unscaledImage[9] << 16
          | unscaledImage[10] << 8
          | unscaledImage[11];

// Construct destination image header

dst = (unsigned char*)malloc(
    HDR_SIZ + (scaledWidth * scaledHeight * PIX_SIZ)
);
memset (dst, 0, HDR_SIZ + (scaledWidth * scaledHeight * PIX_SIZ));

dst[0] = 'P';
dst[1] = 'H';
dst[2] = '2';
dst[3] = '4';

dst[4] = (scaledWidth >> 24) & 0xFF;
dst[5] = (scaledWidth >> 16) & 0xFF;
dst[6] = (scaledWidth >> 8) & 0xFF;
dst[7] = scaledWidth & 0xFF;

dst[8] = (scaledHeight >> 24) & 0xFF;
```

```
dst[9] = (scaledHeight >> 16) & 0xFF;
dst[10] = (scaledHeight >> 8) & 0xFF;
dst[11] = scaledHeight & 0xFF;

// Copy image data

for (dstY = 0; dstY < scaledHeight; dstY++) {
    fY = dstY * unscaledHeight / scaledHeight + yOfs;
    tY = ((dstY + 1) * unscaledHeight + scaledHeight - 1) /
    scaledHeight + yOfs;
    for (dstX = 0; dstX < scaledWidth; dstX++) {
        fX = dstX * unscaledWidth / scaledWidth + xOfs;
        tX = ((dstX + 1) * unscaledWidth + scaledWidth - 1) /
            scaledWidth + xOfs;

        totW = 0;
        totR = 0;
        totG = 0;
        totB = 0;

        for (srcY = fY; srcY < tY; srcY++) {
            wY = min ((srcY+1)*scaledHeight, (dstY+1)*unscaledHeight)
                - max (srcY*scaledHeight, dstY*unscaledHeight);

            for (srcX = fX; srcX < tX; srcX++) {
                wX = min ((srcX+1)*scaledWidth, (dstX+1)*unscaledWidth)
                    - max (srcX*scaledWidth, dstX*unscaledWidth);

                srcOfs = CoorToOfs(srcX, srcY, srcWidth);
                w = wX * wY;

                if (
                    srcX >= 0 && srcY >=0 &&
                    srcX < srcWidth && srcY < srcHeight
                ) {
                    red = unscaledImage[srcOfs];
                    green = unscaledImage[srcOfs + 1];
                    blue = unscaledImage[srcOfs + 2];
                } else {
                    red = fillRed;
                    green = fillGreen;
                    blue = fillBlue;
                }

                totW += w;
                totR += w * red;
                totG += w * green;
                totB += w * blue;
            }
        }

        red = (totR + totW / 2) / totW;
        green = (totG + totW / 2) / totW;
        blue = (totB + totW / 2) / totW;
```

```
            dstOfs = CoorToOfs (dstX, dstY, scaledWidth);
            dst[dstOfs] = red;
            dst[dstOfs + 1] = green;
            dst[dstOfs + 2] = blue;
        }
    }

    return dst;
}

//------------------------------------------------------------------------------

// MAIN LINE

int
main()
{
    long srcX = 100;                // Source X
    long srcY = 100;                // Source Y
    unsigned long srcW = 300;       // Source width
    unsigned long srcH = 400;       // Source height
    unsigned long dstW = 350;       // Destination width
    unsigned long dstH = 480;       // Destination height

    unsigned char *srcImage = ReadImage("src.ph");

    unsigned char *dstImage = ScaleImage (
        srcImage, srcX, srcY, srcW, srcH, dstW, dstH, 166, 202, 240
    );

    WriteImage("dst.ph", dstImage, HDR_SIZ + (dstW * dstH * PIX_SIZ));

    return 0;
}
```