Course IPCis: Image Processing with C#
Chapter C3: The Commented Code of the Filter Project

`using System;` //Home of the base class of all classes "`System.Object`" and of all primitive data types such as `Int32, Int16, double, string`.

`using System.Drawing;` //Home of the "`Graphics`" class and its drawing methods such as `DrawStirng, DrawLine, DrawRectangle, FillClosedCurve` etc.

`using System.Drawing.Imaging;` //Home of the `Bitmap`-class.

```
using System.Windows.Forms;
```
//Home of the "`Form`" class (base class of our main window `Form1)` and its method `Application.Run`.

Entry `public class Form1 : Form`

//We derive our window `public class Form1 : Form`, which the compiler automatically finds in the `System.Windows.Forms` namespace.

```
{ static void Main() { Application.Run( new Form1() ); }
```
//Create an instance of `Form1` and ask the operating system to start it as main window of our program.

```
Brush bbrush = SystemBrushes.ControlText;
```
//Create a reference to an already existing `Brush` object (just a programming shortcut).

```
Bitmap Original, Noise, Lowpass, HighVertical, HighHorizontal, HighGradient;
```
//Six `Bitmap`-objects:
1. row: `Original` = original image, `Noise` = noisy image, `Lowpass` = blurred image
2. row: `HighVertical` = vertical highpass, `HighHorizontal` = horizontal highpass, `HighGradient = SQRT(HighVertical`^{2} `+ HighVertical`^{2}`)`.

`Byte[,] grayarray;` //Raster matrix containing `Original` as monochrome image.
Any gray value is the average of its three correspondent `Original`-RGB-values:
```
grayarray[x,y] = ( Original[x,y].R + Original[x,y].G + Original[x,y].B ) / 3;
```

`Int32 x, y, xSize, ySize, gray;` //Some global integers.

Constructor `public Form1()`

```
MenuItem miRead = new MenuItem( "&Read", new EventHandler( MenuFileRead ) );
```
//Purpose: Open an image file.

```
MenuItem miExit = new MenuItem( "&Exit", new EventHandler( MenuFileExit ) );
```
//Purpose: Leave the program.

```
MenuItem miFile = new MenuItem( "&File", new MenuItem[] { miRead, miExit } );
```
//Complete menu with two entries.

```
Menu = new System.Windows.Forms.MainMenu( new MenuItem[] { miFile } );
```
//Attach the menu to `Form1`.

```
Text = "Filter1";
```
//Blue title bar.
```
SetStyle( ControlStyles.ResizeRedraw, true );
```
//Redirect the `OnResize`-event to the `OnPaint` event handler.
```
Width = 1024;
```
//Initial window size.
```
Height = 800;
```
//Initial window size.
Raise the `OnResize`-event which calls `protected override void OnPaint(...)`.

Event Handler `void MenuFileRead( object obj, EventArgs ea )`

```
{ OpenFileDialog dlg = new OpenFileDialog(); //Call the standard modal OpenFileDialog-box.
```

```
dlg.Filter = "bmp files (*.bmp)|*.bmp|All files (*.*)|*.*" ;
```
//Pre select `*.bmp`-files, but choice of any other file type remains possible.

```
if ( dlg.ShowDialog() != DialogResult.OK ) return;
```
//Forget it, if there was no reasonable user reaction.

```
Original = (Bitmap)Image.FromFile( dlg.FileName );
```
//Read the image using the powerful `Bitmap`-class.

```
if ( Original == null ) return; //Forget it, when no image could be opened.
```

```
Cursor.Current = Cursors.WaitCursor;
```
//Change the mouse pointer to hour-glass indicating that a time consuming process is going to run.

```
xSize = Original.Width; //Use the traditional xSize for image width = no. of columns.
```

```
ySize = Original.Height; //Use the traditional ySize for image height = no. of lines.
```

```
Noise = new Bitmap( xSize, ySize, PixelFormat.Format16bppRgb555 );
```
//Image with 16 bits per pixel.

```
Lowpass = new Bitmap( xSize, ySize, PixelFormat.Format24bppRgb );
```
//Image with 24 bits per pixel = best quality.

```
HighVertical = new Bitmap( xSize, ySize, PixelFormat.Format16bppRgb555 );
```
//Image with 16 bits per pixel.

```
HighHorizontal = new Bitmap( xSize, ySize, PixelFormat.Format16bppRgb555 );
```
//Image with 16 bits per pixel.

```
HighGradient = new Bitmap( xSize, ySize, PixelFormat.Format16bppRgb555 );
```
//Image with 16 bits per pixel.

```
grayarray = new Byte [xSize, ySize];
```
//Raster matrix containing `Original` as monochrome Byte-image.

```
for ( y=0; y < ySize; y++ ) //For any line
for ( x=0; x < xSize; x++ ) //For any column
{ Color color = Original.GetPixel( x, y );  //Read a pixel.
gray = ( color.R + color.G + color.B ) / 3; //Sum up the three colors and divide = averaging.
grayarray[x, y] = (Byte)gray;               //Store the gray value.
```

```
//Version 2 Noise*********************************************************
```

```
Int32 amplitude = 256; //Maximal noise = additional random gray
Int32 half_amplitude = amplitude / 2; //Half noise = 128
Random random = new Random(); //Start the random no. generator
```

```
for ( y=0; y < ySize; y++ ) //For any line
for ( x=0; x < xSize; x++ ) //For any column
```

```
{ gray = grayarray[x, y]; //Read a pixel.
gray += random.Next(amplitude) - half_amplitude; //Add a random gray between -128 and +128
if (gray < 0) gray = 0; else if (gray > 255) gray = 255;
```
//Clip the resulting value to minimum=0 and maximum=256.
```
Noise.SetPixel( x, y, Color.FromArgb( gray, gray, gray ) ); //Store the noisy pixel.
```

```
//Version 3 Lowpass without border handling*****************************
// Version 3: Lowpass without border handling //
```

```
Int32 LowpassSize = 11; //insert an odd size: 3, 5, 7, ..., 29.
```
//Width and height of the filter

```
Int32 MidWeight = 1; //set a positive mid weight: 1, 2, 3, ..., 1000.
```
//Mid of the filter has no special weight yet.

```
Int32 xx, yy, sum, d = LowpassSize/2;
```
//Local indicees: `xx, yy`; local sum = `sum`; half width = half height = `d`;

```
float divisor = (2*d+1)*(2*d+1) + MidWeight - 1; //MidWeight == 1 means no weight
```
//Sum of all weights (normally 11*11 = 121 = no. of elements of this averaging lowpass filter).

```
for ( y=d; y < ySize-d; y++ )
```
//For any line except the upper (`0` to `d-1`) and lower (`ySize-d` to `ySize-1`) rim = half size of the quadratic filter (normally = 5).

```
for ( x=d; x < xSize-d; x++ )
```
//For any column except the left (`0` to `d-1`) and right (`xSize-d` to `xSize-1`) rim = half size of the quadratic filter (normally = 5).

```
sum = (MidWeight-1) * grayarray[x,y]; //extra mid weight
```
//When there is no special `MidWeight`, `sum` starts with `0`, otherwise it starts with the wéighted central gray value.

```
for ( yy=-d; yy <= d; yy++ )
```
//For any y-offset between `-d` and `+d`, where `d` is the rim = the half size of the quadratic filter (normally = 5).

```
for ( xx=-d; xx <= d; xx++ )
```
//For any x-offset between `-d` and `+d`, where `d` is the rim = half size of the quadratic filter (normally = 5).

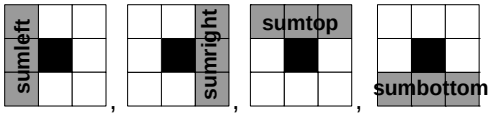
```
sum += grayarray[x+xx,y+yy];
```
//Add up the gray values of the `11*11` neighbourhood.

```
gray = Convert.ToByte( (float)sum / divisor );
```
//Divide by the sum of weights and round to `Byte`.

```
Lowpass.SetPixel( x, y, Color.FromArgb( gray, gray, gray ) );
```
//Store the pixel.

```
//Version 4 HighHorizontal + HighVertical + Highgradient***************
```

```
Int32 sumleft,
sumright,
sumtop,
sumbottom,
diff;
```




```
diff = Abs(sumleft-sumright)
```
or
```
diff = Abs(sumtop-sumbottom)
```

```
for ( y=1; y < ySize-1; y++ )
```
//For any line except the 1st and last ones.
```
for ( x=1; x < xSize-1; x++ )
```
//For any column except the 1st and last ones.

```
sumleft = grayarray[x-1,y] + grayarray[x-1,y-1] + grayarray[x-1,y+1];
```
//Add the 3 left neighbours.

```
sumright = grayarray[x+1,y] + grayarray[x+1,y-1] + grayarray[x+1,y+1];
```
//Add the 3 right neighbours.

```
sumtop = grayarray[x,y-1] + grayarray[x-1,y-1] + grayarray[x+1,y-1];
```
//Add the 3 upper neighbours.

```
sumbottom = grayarray[x,y+1] + grayarray[x-1,y+1] + grayarray[x+1,y+1];
```
//Add the 3 lower neighbours.

```
diff = Math.Abs( sumleft - sumright );
```
//Subtract left minus right, diff must be positive

```
if ( diff > 255 ) HighHorizontal.SetPixel( x,y,Color.FromArgb( 255, 255, 255 ) );
```
//Max. allowed difference = 255.

```
else HighHorizontal.SetPixel( x,y,Color.FromArgb( diff, diff, diff ) );
```
//Store pixel.

```
diff = sumtop - sumbottom; if ( diff < 0 ) diff *= -1;
```
//Subtract top minus bottom.

```
if ( diff > 255 ) HighVertical.SetPixel( x,y,Color.FromArgb( 255, 255, 255 ) );
```
//Max. allowed difference = 255.

```
else HighVertical.SetPixel( x,y,Color.FromArgb( diff, diff, diff ) );
```
//Store pixel.

```
Double diff_h = sumleft - sumright;
```
//Convert difference to type `Double` as requested by `Math.Sqrt`.

```
Double diff_v = sumtop - sumbottom;
```
//Convert difference to type `Double` as requested by `Math.Sqrt`.

```
diff = Convert.ToInt32( Math.Sqrt( diff_h * diff_h + diff_v * diff_v ) );
```
//Pythagorean theorem and rounding.

```
if ( diff > 255 ) HighGradient.SetPixel( x,y,Color.FromArgb( 255, 255, 255 ) );
```
//Max. allowed gradient = 255.

```
else HighGradient.SetPixel( x,y,Color.FromArgb( diff, diff, diff ) );
```
//Store pixel.

```
Cursors.Current = Cursors.Arrow;
```
//Reset the mouse cursor to its default shape.

```
Invalidate();
```
//Call `protected override void OnPaint(...)` and show the 6 images.

Event Handler `void MenuFileExit( object obj, EventArgs ea )`

```
{ Application.Exit(); }
```
//Kill the current instance of program `filter1`.

Event Handler `protected override void OnPaint( PaintEventArgs e )`

```
Graphics g = e.Graphics;
```
//Get the current device context.

```
if ( Original == null )
  { g.DrawString( "Open an Image File !", Font, bbrush, 0, 0 ); return; }
```
//Inform the user what to do, if there is no image at all.

```
Rectangle cr = ClientRectangle;
```
//Get the size of the whole client area.

```
g.DrawImage( Original , 0 , 0 , cr.Width/3, cr.Height/2 );
```
//Upper left corner.

```
g.DrawImage( Noise , cr.Width/3 , 0 , cr.Width/3, cr.Height/2 );
```
//In the mid of the upper row.

```
g.DrawImage( Lowpass , (2*cr.Width)/3, 0 , cr.Width/3, cr.Height/2 );
```
//Upper right corner.

```
g.DrawImage( HighHorizontal, 0 , cr.Height/2, cr.Width/3, cr.Height/2 );
```
//Lower left corner.

```
g.DrawImage( HighVertical , cr.Width/3 , cr.Height/2, cr.Width/3, cr.Height/2 );
```
//In the mid of the lower row.

```
g.DrawImage( HighGradient , (2*cr.Width)/3, cr.Height/2, cr.Width/3, cr.Height/2
);
```
//Lower right corner.