```
private void Niblack_BTN_Click(object sender, EventArgs e)
        {
            Binary_Image = Gray_Image.CopyBlank();

            int w = 15;            /* window size */
            int w_half = w >> 1;
            int num_rows, num_cols;
            int win_count = w * w;     /* number of pixels in the filtering window */
            int ir, ic;
            int iwr, iwc;
            int r_begin, r_end;      /* vertical limits of the filtering operation */
            int c_begin, c_end;      /* horizontal limits of the filtering operation */
            int wr_begin, wr_end;      /* vertical limits of the filtering window */
            int wc_begin, wc_end;      /* horizontal limits of the filtering window */
            int gray_val;
            int sum, sum_sq;      /* temp variables used in the calculation of local mean
and variances */
            int threshold = 0;
            double local_mean;      /* gray level mean in a particular window position */
            double local_var;      /* gray level variance in a particular window position
*/
            /* Determines how much of the foreground object edges that are taken as a
part of the object */
            double k_value = 0.2;  /* Niblack recommends K_VALUE = -0.2 for images with
black foreground objects, and K_VALUE = +0.2 for images with white foreground objects. */

            num_rows = Gray_Image.Height;
            num_cols = Gray_Image.Width;

            /*
               Determine the limits of the filtering operation. Pixels
               in the output image outside these limits are set to 0.
             */
            r_begin = w_half;
            r_end = num_rows - w_half;
            c_begin = w_half;
            c_end = num_cols - w_half;

            /* Initialize the vertical limits of the filtering window */
            wr_begin = 0;
            wr_end = w;

            /* For each image row */
            for (ir = r_begin; ir < r_end; ir++)
            {
                /* Initialize the horizontal limits of the filtering window */
                wc_begin = 0;
                wc_end = w;

                /* For each image column */
                for (ic = c_begin; ic < c_end; ic++)
                {
                    sum = sum_sq = 0;
                    /* For each window row */
                    for (iwr = wr_begin; iwr < wr_end; iwr++)
                    {
                        /* For each window column */
                        for (iwc = wc_begin; iwc < wc_end; iwc++)
```

```
                        {
                            gray_val = Gray_Image.Data[iwr, iwc, 0];
                            sum += gray_val;
                            sum_sq += gray_val * gray_val;
                        }
                    }

                    /* Calculate the local mean and variance */
                    local_mean = sum / (double)win_count;
                    local_var = (sum_sq / (double)win_count) - local_mean * local_mean;

                    /* Calculate local threshold */
                    threshold = (int)(local_mean + k_value * Math.Sqrt(local_var));

                    /* Determine the output pixel value */
                    Binary_Image.Data[ir, ic, 0] =
                     ((Gray_Image.Data[ir, ic, 0] > threshold) ? MAX_BRIGHTNESS :
MIN_BRIGHTNESS);

                    /* Update the horizontal limits of the filtering window */
                    wc_begin++;
                    wc_end++;
                }

                /* Update the vertical limits of the filtering window */
                wr_begin++;
                wr_end++;
            }

            image_IMGBXEMGU.Image = Binary_Image;
        }
```