Lab 3

Use two images for each operation to do the following operations through write functions:

1. Image translation
2. Image rotation
3. Shear operations(vertical and horizontal) respectively
4. Smoothing with 3×3, 5×5 windows using averaging and median filters respectively

Algorithm:

1. Image translation

    Input: an unsigned char array that stores the source image data, translate units (x,y)

    Output: an unsigned char array that stores the output image data

```
outimage. width =inimage.width +abs(x)
outimage.height=inimage.height+abs(y)

Let every pixel in outimage become 0
    if ty <= 0 sy = 0; //If the translate unit is negative, than start with 0
    if tx <= 0 sx = 0;

    for (i = sy; i < sy + image->Height; i++) {
        for (j = sx; j < sx + image->Width; j++) {
            outimage[i][j] = inimage[(i-sy)][(j-sx)];
     }
    }
    return outImage;
```

2. Image rotation

    Input: an unsigned char array that stores the source image data, rotation degree θ

    Output: an unsigned char array that stores the output image data

```
outimage. width = abs(inimage. Width* sin(θ)+ inimage.height* cos(θ))
outimage.height= abs(inimage. Width* cos(θ) + inimage.height* sin(θ))

Let every pixel in outimage become 0
    for (i = 0; i <inimage->Height; i++) {
        for (j = 0; j < inimage->Width; j++) {// Find the project point coordinates on the
outimage
            x = round(i * sin(θ)   - j * cos(θ)+(float)(image->Width*cos(θ))); // Add the
                                                                    //offset
            y = round(i * cos(θ) + j * sin(θ));
            outimage [x][y] = inimage [i][j];
        }
    }

    // Fill the empty pixels with 3×3 average interpolation
    for (i = 1; i < outImage->Height-1; i++) {
        for (j = 1; j < outImage->Width-1; j++) {
            if (tempout[i * outImage->Width + j] == 0) {// If the pixel is empty then conduct
interpolation
                sum = 0;
                temp = 0;
                for (p = -1; p < 2; p++) {
                    for (q = -1; q < 2; q++) {
                        tempArr[temp++] = outimage[(i + p)][ (j + q)];
                        sum = sum + outimage[(i + p)][ (j + q)];

                    }
                }
                tempout[i] [j] = sum / 8; // Fill the empty pixel with the average value of its
neighbours
            }
        }
    }
    return outimage
```

3. Shear operations(vertical and horizontal) respectively
   1) Veritical shear
      Input: an unsigned char array that stores the source image data, shear degree θ
      Output: an unsigned char array that stores the output image data

```
outimage.width= inimage.width
outimage.height= inimage. height +abs(image. width * tan(θ))
Let every pixel in outimage become 0

 for (i = 0; i < image->Height; i++) {
     for (j = 0; j < image->Width; j++) {
         x = round(i- tan(θ)*j+inimage.Width* tan(θ));
         y = j;
         outimage[x][y] = inimage[i][j];
     }
 }
```

      Conduction the average interpolation same as the rotation.
      return outimage
   2) Horizontal shear
      Input: an unsigned char array that stores the source image data, shear degree θ
      Output: an unsigned char array that stores the output image data

```
outimage.width= inimage.width+abs(image.height * tan(θ))
outimage.height= inimage.height

 Let every pixel in outimage become 0
for (i = 0; i < image->Height; i++) {
     for (j = 0; j < image->Width; j++) {
         x =i;
         y =round(- tan(θ)*i+j+inimage.Width* tan(θ));
         outimage[x][y] = inimage[i][j];
```

```
            }
        }
    Conduction the average interpolation same as the rotation.
    return outimage
```

4. Smoothing with 3×3, and 5×5 windows using averaging and median filters respectively
   1) averaging filter

      Input: an unsigned char array that stores the source image data, the size of mask

      Output: an unsigned char array that stores the output image data

```
for (i = size / 2; i < inimage.Width -size / 2; i++) {
    for (j = size / 2; j < inimage.Height - size / 2; j++) {
        temp = 0;
        for (p = -1; p < size-1; p++) {
            for (q = -1; q < size-1; q++) {
                temp = temp+tempin[(i + p)][(j + q)];
            }
        }
        outimage[i ][ j] = temp /(size * size);
    }
}
return outImage;
```

   2) Median filter

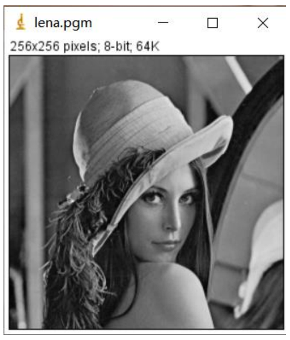
      Input: an unsigned char array that stores the source image data, the size of mask

      Output: an unsigned char array that stores the output image data

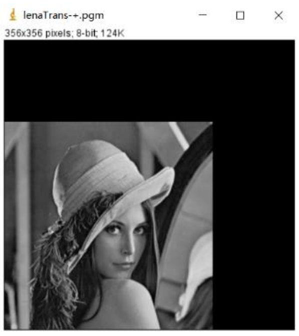
```
for (i = size / 2; i < inimage.Width - size / 2; i++) {
    for (j = size / 2; j < inimage.Height - size / 2; j++) {
        temp = 0;
        for (p = -1; p < size - 1; p++) {
            for (q = -1; q < size - 1; q++) {
                tempArr[temp++]=inimage[(i + p)][(j + q)];
            }
        }
        outimage[i ][j] = Median(tempArr,size*size);
    }
}
return outimage
```

Results (compare the results with the original image):

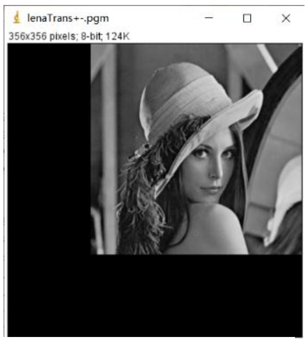
1. Image translation



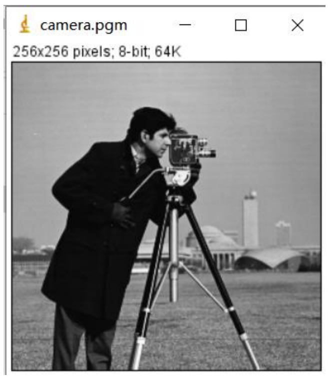
Source

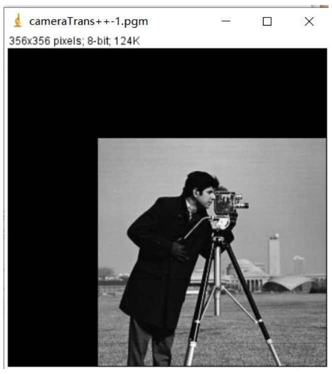


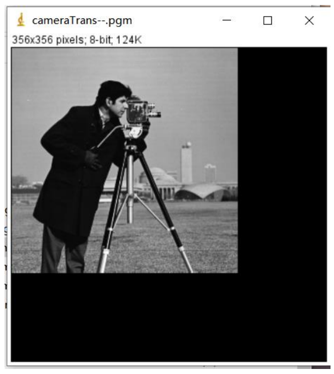
(-100,+100)



(100,-100)

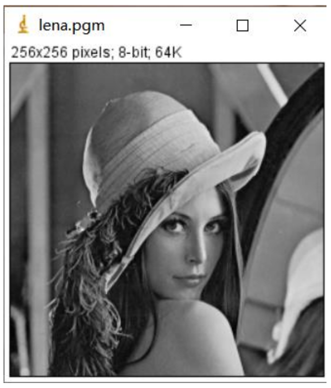


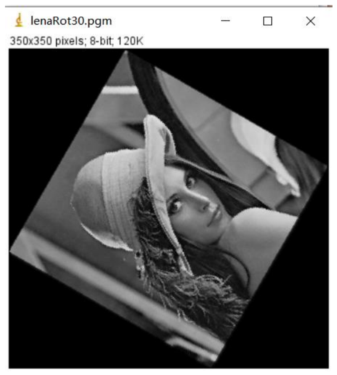


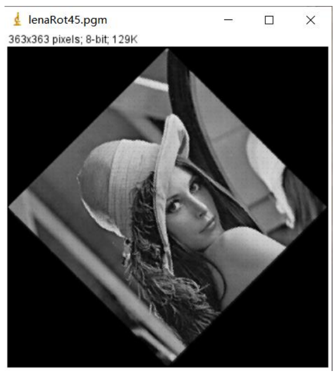


Source (100,100) (-100,-100)

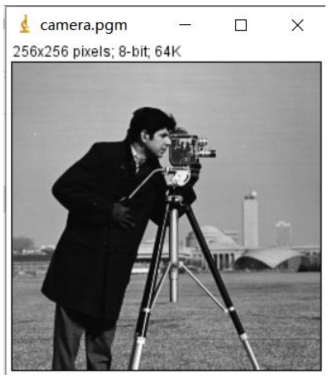
2. Image rotation

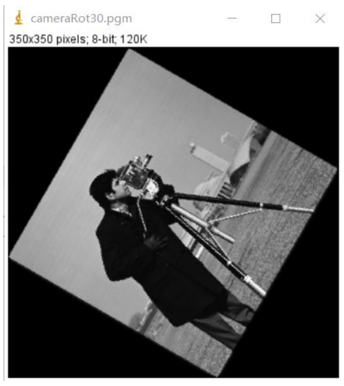


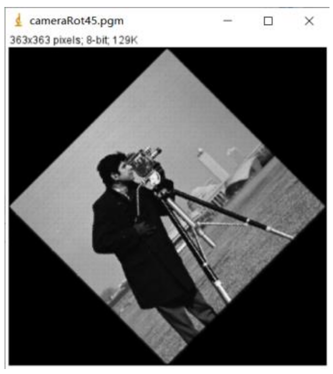




Source Rotate 30 Rotate 45

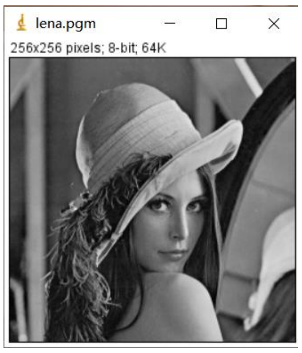


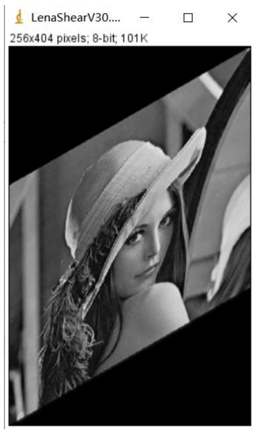


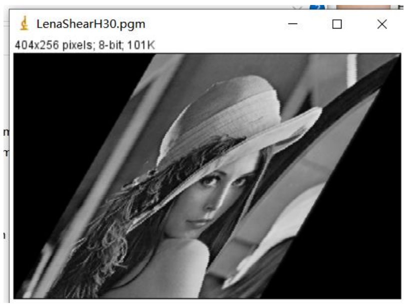


Source Rotate 30 Rotate 45

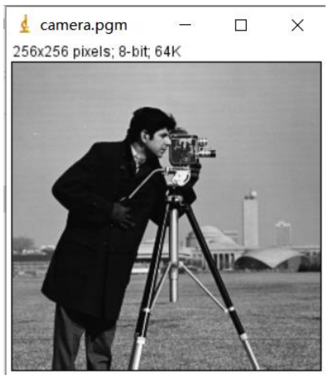
3. Shear operations(vertical and horizontal) respectively

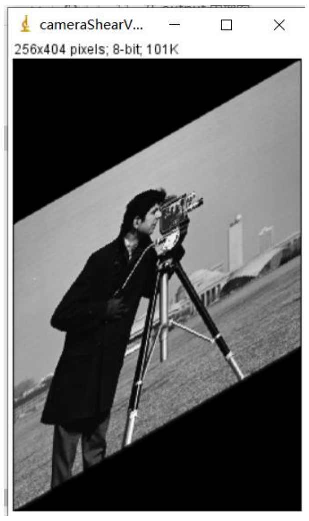


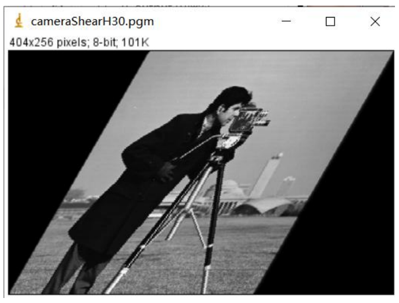




Source vertical 30 horizontal 30

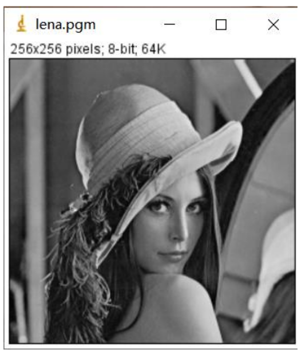


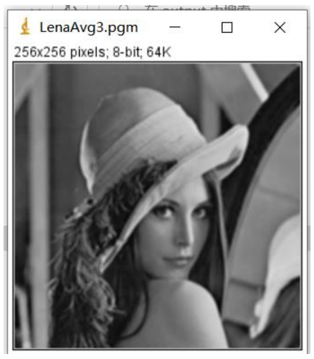


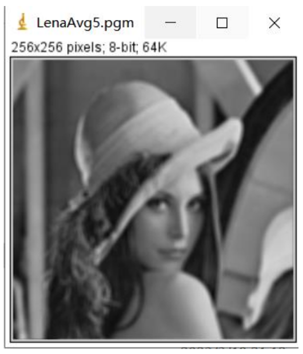


Source vertical 30 horizontal 30

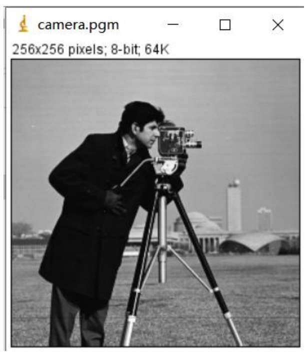
4. Smoothing with 3×3, and 5×5 windows using averaging and median filters respectively
1) Averaging filter

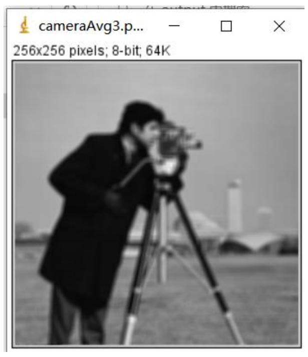


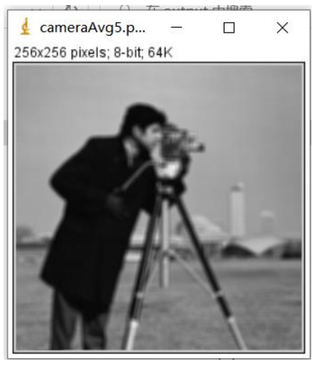




Source 3×3 5×5

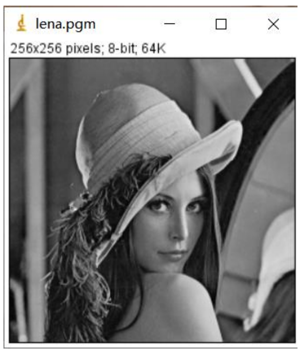


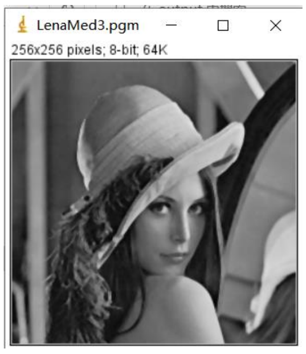


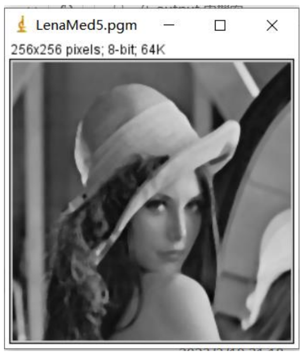


Source 3×3 5×5

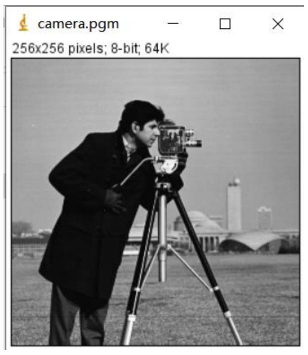
2) Median filter



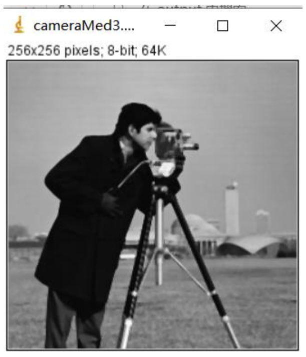




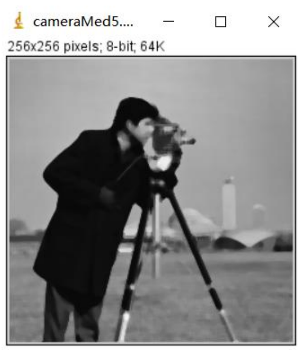
Source 3×3 5×5



Source



3×3



5×5

Discussion:

1. Image translation

   Should check whether the translate unit is positive or not, if the unit is negative, the projection of outimage array should start with zero, else than start with the unit values. This action can leave an empty area of the outimage

2. Image rotation

   When conducting interpolation of the outimage, here I choose to use the average interpolation. At first I set the interpolate value as the average of 9 neighbors, but the result is not that good because the value of the central pixel is 0. Then I set the value as the average of 8 neighbours, then the performance is better than before.

3. Shear operations(vertical and horizontal) respectively

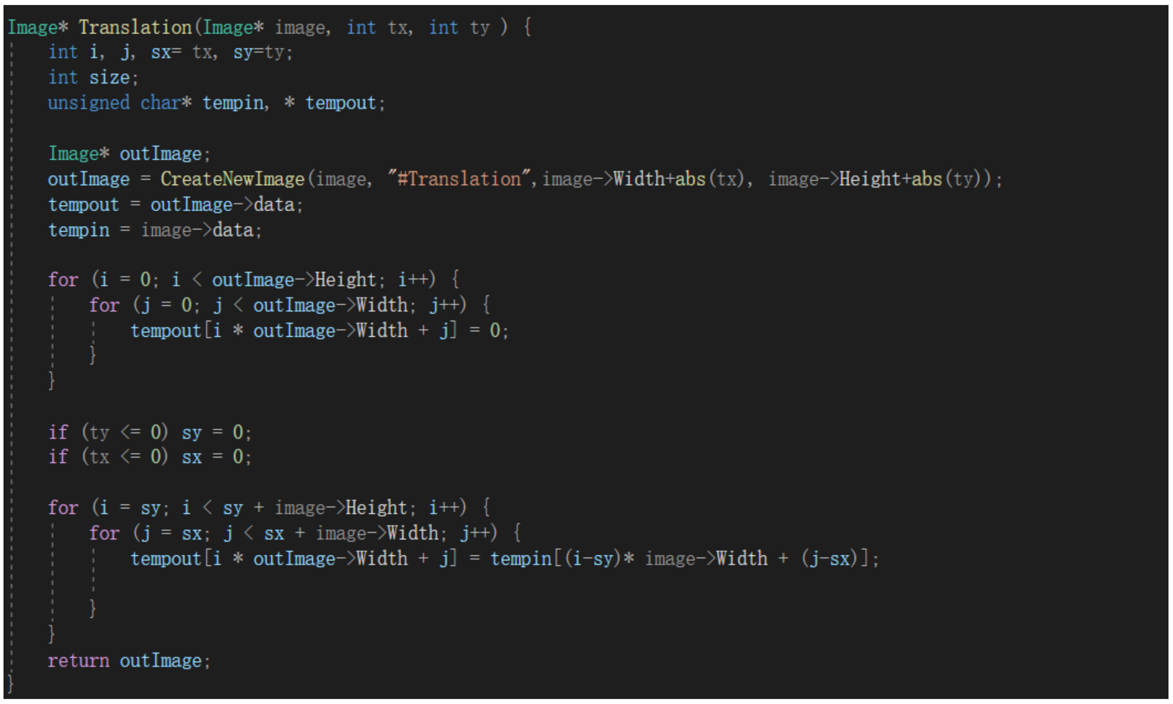
   I think the most important step is to calculate an accurate width and height of the output image. And also different direction of shearing has a different conduction

4. Smoothing with 3×3, and 5×5 windows using averaging and median filters respectively

   The drawback of this method is that the edges of the output image cannot have value.

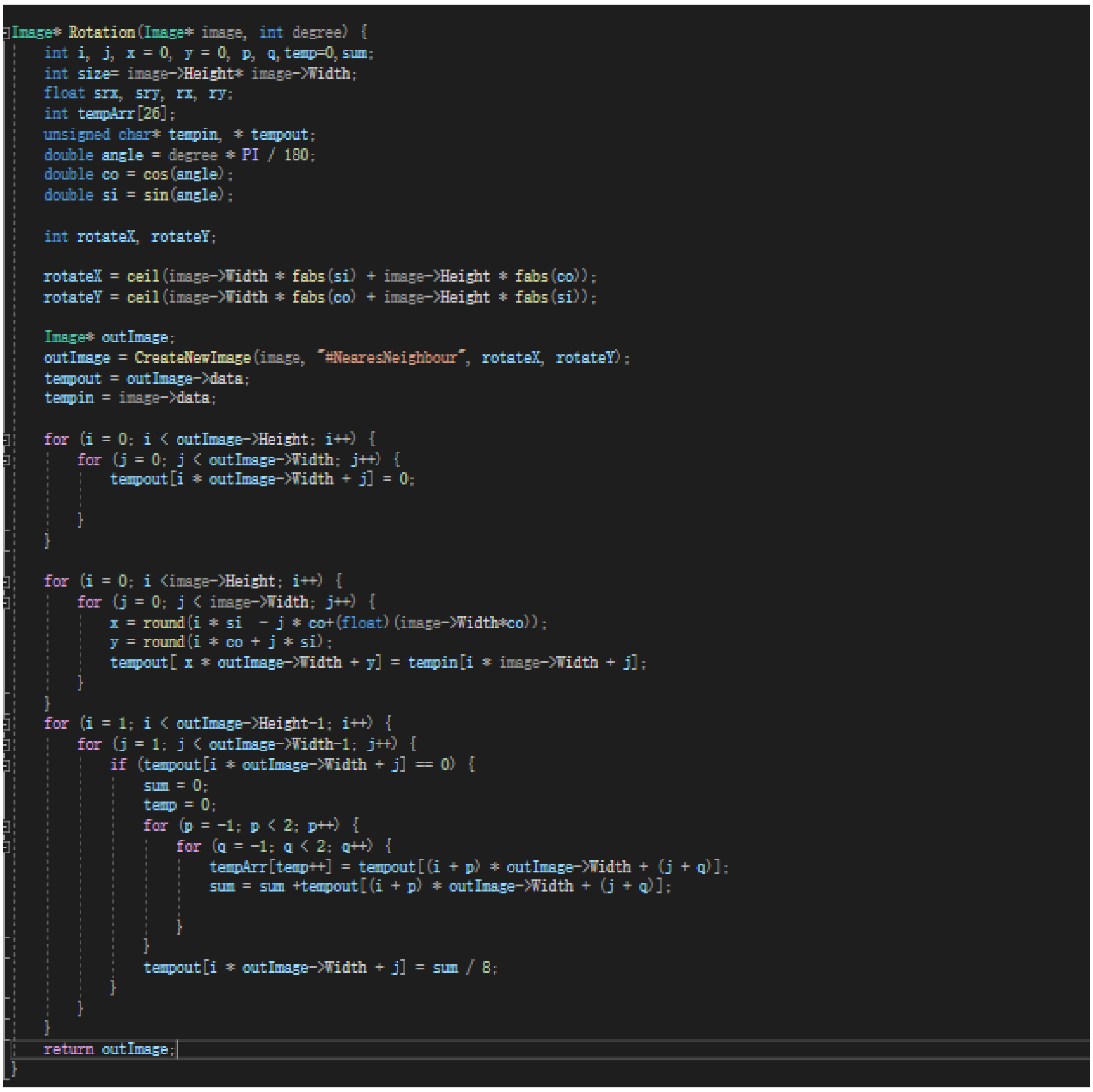
Codes:

1. Image translation



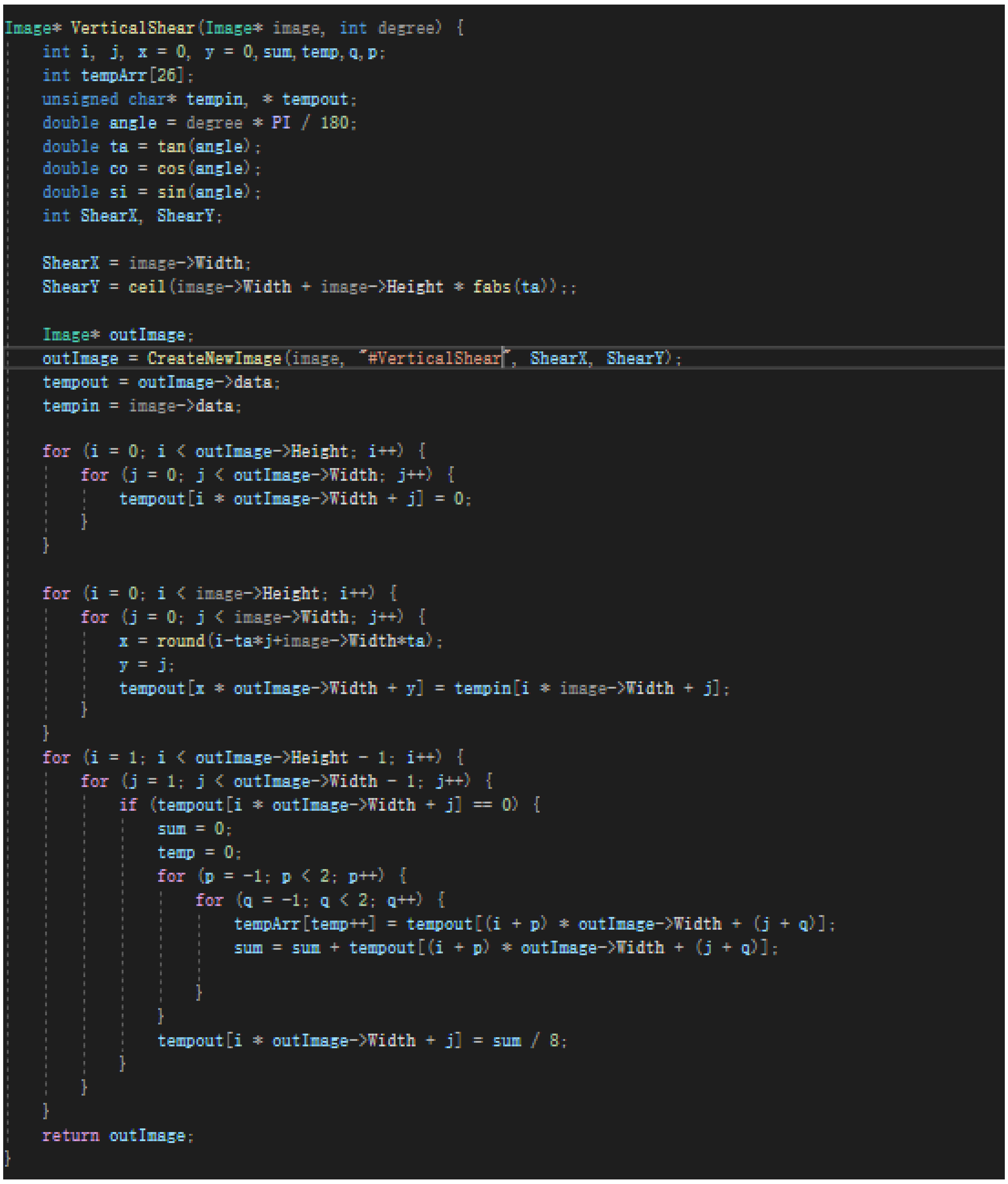
```
Image* Translation(Image* image, int tx, int ty ) {
    int i, j, sx= tx, sy=ty;
    int size;
    unsigned char* tempin, * tempout;

    Image* outImage;
    outImage = CreateNewImage(image, "#Translation",image->Width+abs(tx), image->Height+abs(ty));
    tempout = outImage->data;
    tempin = image->data;

    for (i = 0; i < outImage->Height; i++) {
        for (j = 0; j < outImage->Width; j++) {
            tempout[i * outImage->Width + j] = 0;
        }
    }

    if (ty <= 0) sy = 0;
    if (tx <= 0) sx = 0;

    for (i = sy; i < sy + image->Height; i++) {
        for (j = sx; j < sx + image->Width; j++) {
            tempout[i * outImage->Width + j] = tempin[(i-sy)* image->Width + (j-sx)];

        }
    }
    return outImage;
}
```

2. Image rotation



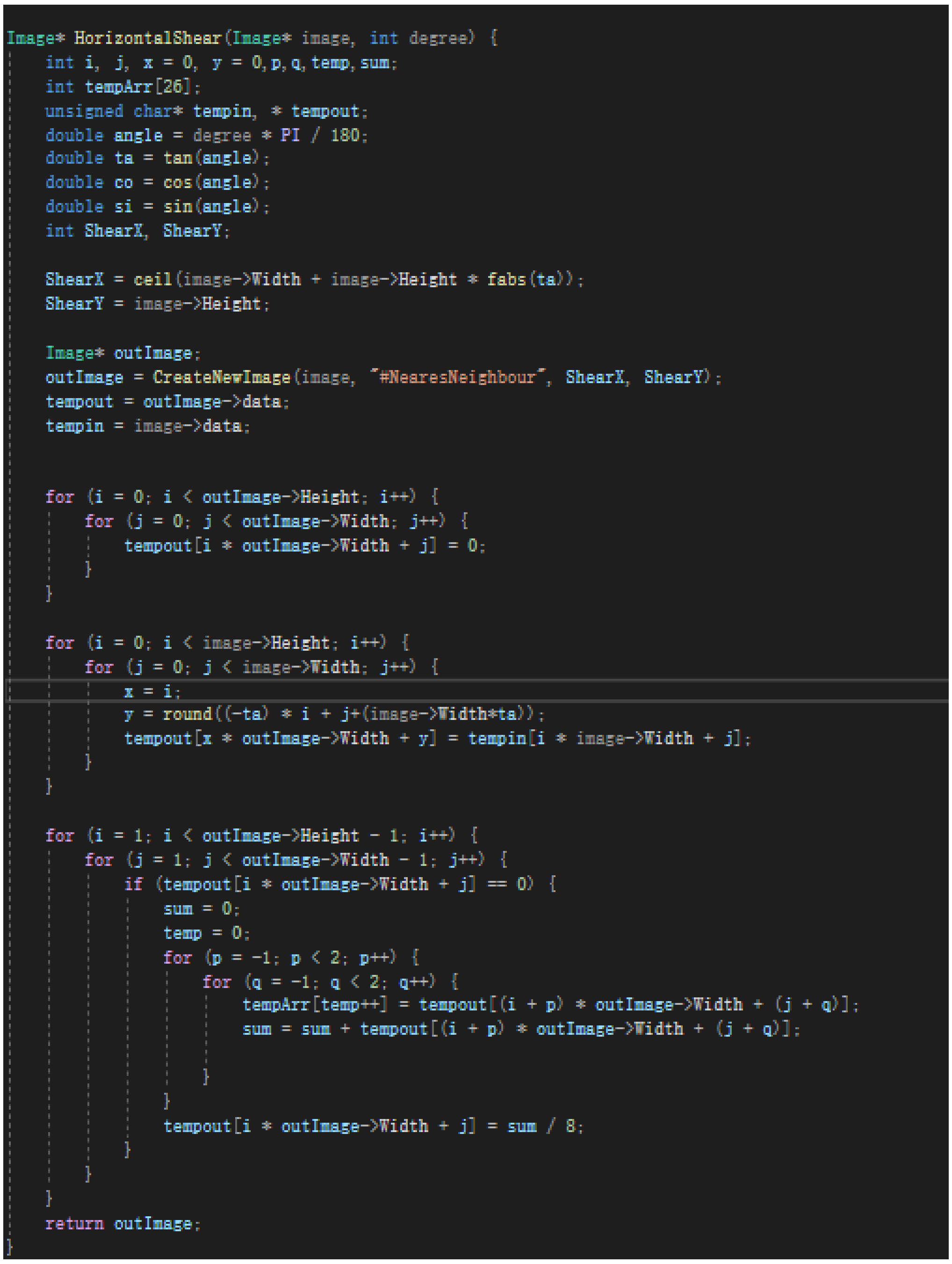
```
Image* Rotation(Image* image, int degree) {
    int i, j, x = 0, y = 0, p, q,temp=0,sum;
    int size= image->Height* image->Width;
    float srx, sry, rx, ry;
    int tempArr[26];
    unsigned char* tempin, * tempout;
    double angle = degree * PI / 180;
    double co = cos(angle);
    double si = sin(angle);

    int rotateX, rotateY;

    rotateX = ceil(image->Width * fabs(si) + image->Height * fabs(co));
    rotateY = ceil(image->Width * fabs(co) + image->Height * fabs(si));

    Image* outImage;
    outImage = CreateNewImage(image, "#NearesNeighbour", rotateX, rotateY);
    tempout = outImage->data;
    tempin = image->data;

    for (i = 0; i < outImage->Height; i++) {
        for (j = 0; j < outImage->Width; j++) {
            tempout[i * outImage->Width + j] = 0;

        }
    }

    for (i = 0; i <image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            x = round(i * si  - j * co+(float)(image->Width*co));
            y = round(i * co + j * si);
            tempout[ x * outImage->Width + y] = tempin[i * image->Width + j];
        }
    }
    for (i = 1; i < outImage->Height-1; i++) {
        for (j = 1; j < outImage->Width-1; j++) {
            if (tempout[i * outImage->Width + j] == 0) {
                sum = 0;
                temp = 0;
                for (p = -1; p < 2; p++) {
                    for (q = -1; q < 2; q++) {
                        tempArr[temp++] = tempout[(i + p) * outImage->Width + (j + q)];
                        sum = sum +tempout[(i + p) * outImage->Width + (j + q)];

                    }
                }
                tempout[i * outImage->Width + j] = sum / 8;
            }
        }
    }
    return outImage;
}
```

3. Shear operations(vertical and horizontal) respectively
    1) Vertical



```
Image* VerticalShear(Image* image, int degree) {
    int i, j, x = 0, y = 0, sum, temp, q, p;
    int tempArr[26];
    unsigned char* tempin, * tempout;
    double angle = degree * PI / 180;
    double ta = tan(angle);
    double co = cos(angle);
    double si = sin(angle);
    int ShearX, ShearY;

    ShearX = image->Width;
    ShearY = ceil(image->Width + image->Height * fabs(ta));;

    Image* outImage;
    outImage = CreateNewImage(image, "#VerticalShear", ShearX, ShearY);
    tempout = outImage->data;
    tempin = image->data;

    for (i = 0; i < outImage->Height; i++) {
        for (j = 0; j < outImage->Width; j++) {
            tempout[i * outImage->Width + j] = 0;
        }
    }

    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            x = round(i-ta*j+image->Width*ta);
            y = j;
            tempout[x * outImage->Width + y] = tempin[i * image->Width + j];
        }
    }
    for (i = 1; i < outImage->Height - 1; i++) {
        for (j = 1; j < outImage->Width - 1; j++) {
            if (tempout[i * outImage->Width + j] == 0) {
                sum = 0;
                temp = 0;
                for (p = -1; p < 2; p++) {
                    for (q = -1; q < 2; q++) {
                        tempArr[temp++] = tempout[(i + p) * outImage->Width + (j + q)];
                        sum = sum + tempout[(i + p) * outImage->Width + (j + q)];

                    }
                }
                tempout[i * outImage->Width + j] = sum / 8;
            }
        }
    }
    return outImage;
}
```

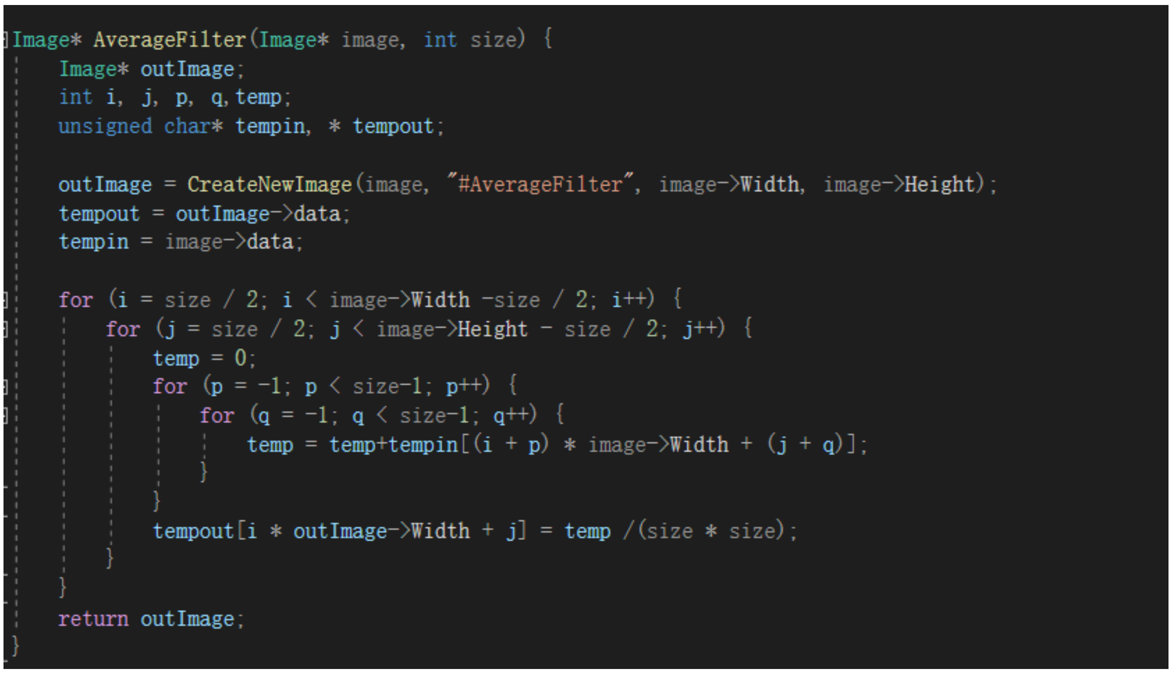
2) Horizontal



```
Image* HorizontalShear(Image* image, int degree) {
    int i, j, x = 0, y = 0,p,q,temp,sum;
    int tempArr[26];
    unsigned char* tempin, * tempout;
    double angle = degree * PI / 180;
    double ta = tan(angle);
    double co = cos(angle);
    double si = sin(angle);
    int ShearX, ShearY;

    ShearX = ceil(image->Width + image->Height * fabs(ta));
    ShearY = image->Height;

    Image* outImage;
    outImage = CreateNewImage(image, "#NearesNeighbour", ShearX, ShearY);
    tempout = outImage->data;
    tempin = image->data;


    for (i = 0; i < outImage->Height; i++) {
        for (j = 0; j < outImage->Width; j++) {
            tempout[i * outImage->Width + j] = 0;
        }
    }

    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            x = i;
            y = round((-ta) * i + j+(image->Width*ta));
            tempout[x * outImage->Width + y] = tempin[i * image->Width + j];
        }
    }

    for (i = 1; i < outImage->Height - 1; i++) {
        for (j = 1; j < outImage->Width - 1; j++) {
            if (tempout[i * outImage->Width + j] == 0) {
                sum = 0;
                temp = 0;
                for (p = -1; p < 2; p++) {
                    for (q = -1; q < 2; q++) {
                        tempArr[temp++] = tempout[(i + p) * outImage->Width + (j + q)];
                        sum = sum + tempout[(i + p) * outImage->Width + (j + q)];

                    }
                }
                tempout[i * outImage->Width + j] = sum / 8;
            }
        }
    }
    return outImage;
}
```

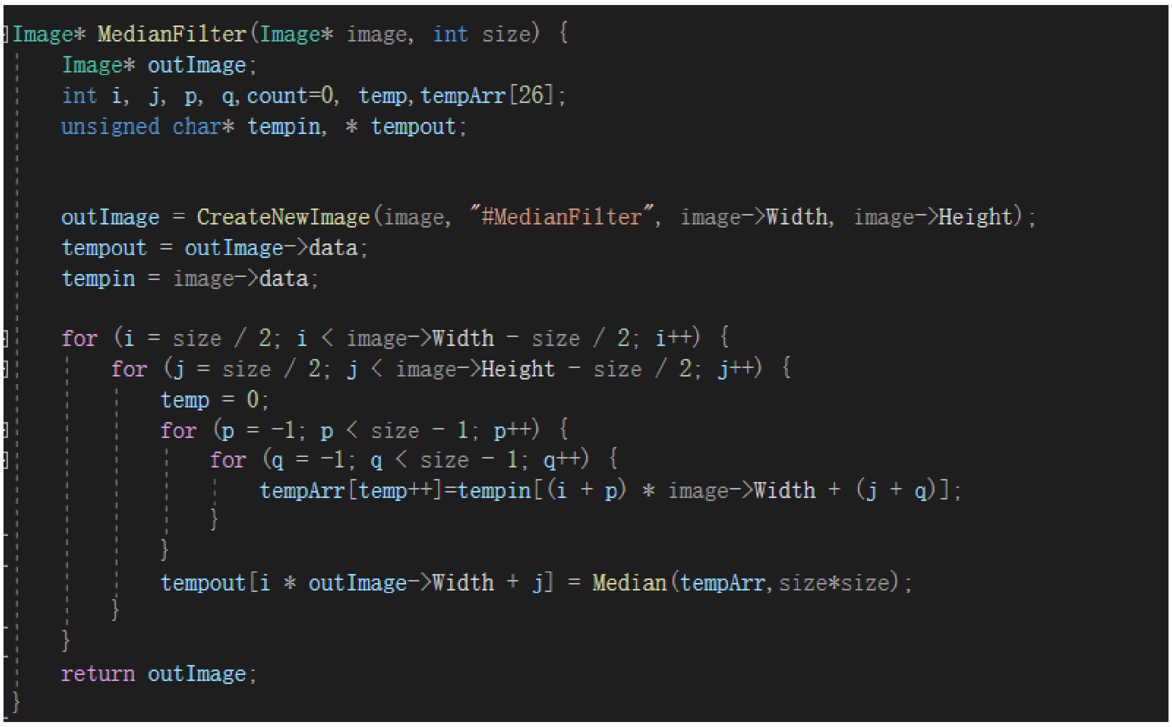
4. Smoothing with 3×3, and 5×5 windows using averaging and median filters respectively

1） Averaging filter

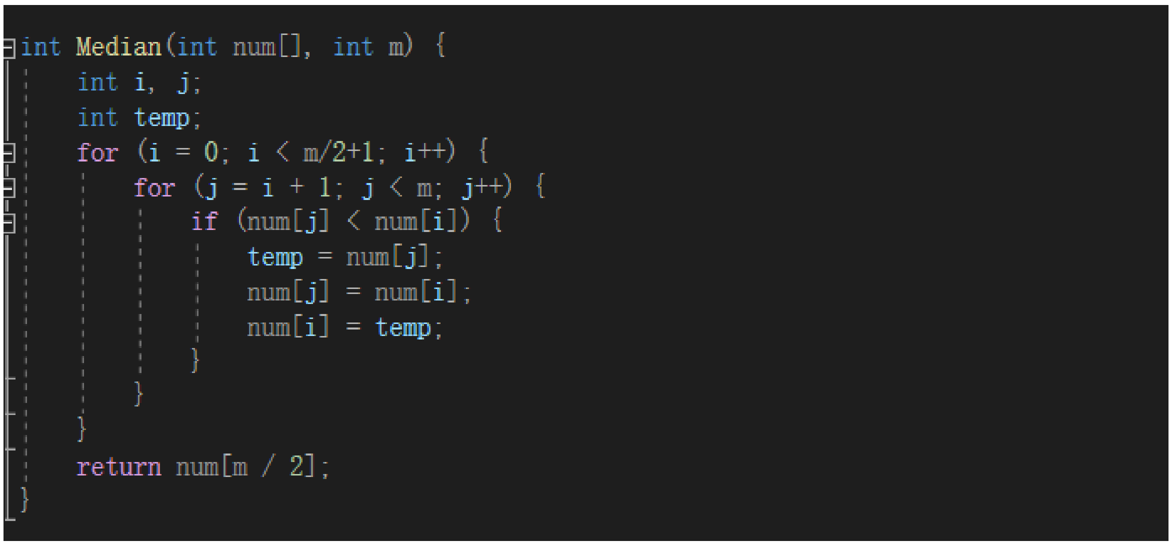


```
Image* AverageFilter(Image* image, int size) {
    Image* outImage;
    int i, j, p, q,temp;
    unsigned char* tempin, * tempout;

    outImage = CreateNewImage(image, "#AverageFilter", image->Width, image->Height);
    tempout = outImage->data;
    tempin = image->data;

    for (i = size / 2; i < image->Width -size / 2; i++) {
        for (j = size / 2; j < image->Height - size / 2; j++) {
            temp = 0;
            for (p = -1; p < size-1; p++) {
                for (q = -1; q < size-1; q++) {
                    temp = temp+tempin[(i + p) * image->Width + (j + q)];
                }
            }
            tempout[i * outImage->Width + j] = temp /(size * size);
        }
    }
    return outImage;
}
```

2) Median Filter



```
Image* MedianFilter(Image* image, int size) {
    Image* outImage;
    int i, j, p, q,count=0, temp,tempArr[26];
    unsigned char* tempin, * tempout;


    outImage = CreateNewImage(image, "#MedianFilter", image->Width, image->Height);
    tempout = outImage->data;
    tempin = image->data;

    for (i = size / 2; i < image->Width - size / 2; i++) {
        for (j = size / 2; j < image->Height - size / 2; j++) {
            temp = 0;
            for (p = -1; p < size - 1; p++) {
                for (q = -1; q < size - 1; q++) {
                    tempArr[temp++]=tempin[(i + p) * image->Width + (j + q)];
                }
            }
            tempout[i * outImage->Width + j] = Median(tempArr,size*size);
        }
    }
    return outImage;
}
```



```c
int Median(int num[], int m) {
    int i, j;
    int temp;
    for (i = 0; i < m/2+1; i++) {
        for (j = i + 1; j < m; j++) {
            if (num[j] < num[i]) {
                temp = num[j];
                num[j] = num[i];
                num[i] = temp;
            }
        }
    }
    return num[m / 2];
}
```