Project 3 (C++): You are to implement the three image enhancement methods taught in class: (1) 3x3 averaging, (2) 3x3 median filter, and (3) 3x3 Gaussian filter.

What you need to do:
1) You will be given two data files (img1 and img2) and a mask for the Gaussian filter.
2) Illustration: (You may hand draw the illustration on a paper or use computer drawing.)
  - On an 8 X 11 paper, draw a 2-D array of img1 with mirror framed (numRows+2) by (numCols+2) on the upper left of the paper; and draw another 2D array (without values) next to it on the upper right of the paper. This pair of arrays is for the illustration of 3x3 averaging.
  - Under the first pair, draw two more pairs of arrays as the same as the first pair. The second pair is for the illustration of 3x3 median filter, and the third pair is for the illustration of 3x3 Gaussian filter. Draw the Gaussian mask under the third pair.
  - Manually computes the 3x3 averaging for every pixel inside of frame of the array on the left of first pair, and write the averaging result on the inside frame of the empty array on the right of the first pair.
  - Manually computes the 3x3 median filter for every pixel inside of frame of the array on the left of second pair, and write the median filter result on the inside frame of the empty array to the right of the second pair.
  - Manually computes the 3x3 Gaussian filter for every pixel inside of frame of the array on the left of third pair, and write the Gaussian filter result on the inside frame of the empty array to the right of the third pair.

** Include your drawing in the Illustration section of your hard copy (after the cover page). If you hand draw the illustration, take a snap shot of it.

2) Implement your program according to the specs below (-6 if you are not follow the specs) and debug your program until your program passes compilation.
3) Run and debug your program with img1 until the results are the same as the three pairs on your drawing.
4) Run your program with img2.
5) Before you submit, pay attention to all your outputs to see if all outputs fit within the pages. No wrapped around.

==================================

Include in your hard copy (pdf file)
  - a cover page (include only algorithm steps in main () function, -1 otherwise.)
  - your illustration. (-2 if omitted.)
  - source code
  - AvgFile for img1 // with meaningful caption
  - MedianFile img1 // with meaningful caption
  - GaussFile for img1 // with meaningful caption
  - logFile for img1 // with meaningful caption //print 4 pages if more.
  - AvgFile for img2 // with meaningful caption
  - MedianFile img2 // with meaningful caption
  - GaussFile for img2 // with meaningful caption
  - logFile for img2 // with meaningful caption //print 4 pages if more.

****

Language: C++

****

Project name: Three image enhancement methods: 3x3 Averaging, 3x3 Median Filter, and 3x3 Gaussian filter

Project points: 12 pts

Due Date: Soft copy (*.zip) and hard copies (*.pdf):

(13/12) +1 early submission: 9/23/2024 Monday before midnight
(12/12) on time: 9/26/2024 Thursday before midnight
(-12/12) non-submission: 9/26/2024 Thursday after midnight

*** Name your soft copy and hard copy files using the naming convention given in Project Submission Requirements; otherwise, your submission will be rejected.

*** All on-line submission MUST include Soft copy (*.zip) and hard copy (*.pdf) in the same email attachments with correct email subject as below; otherwise, your submission will be rejected.

Email subject: (CV) first name last name <Project 3: Three image enhancement methods (C++)>

*** Inside the email body include your answer to the 4 questions. Optional screen recording if you wish.

============= Specs ====== -2 if you hard-code file name =================

****
I. Input files:

a) inFile (argv [1]): A txt file representing a grey-scale image with image header.

b) maskFile (argv [2]): a mask for convolution, with the following format:

MaskRows MaskCols MaskMin MaskMax,
follow by MaskRows by MaskCols of pixel values
For example, a 3 by 3 mask may be
3 3 1 4
1 2 1
2 4 2
1 2 1

c) a threshold value (argv [3]) // USE 22

****
II. Output files:

1) AvgFile (argv [4]): input image, the result of 3x3 averaging, and the binary threshold result.
2) MedianFile (argv [5]): input image, the result of 3x3 median filter and the binary threshold result.
3) GaussFile (argv [6]): input image, the result of 3x3 Gaussian filter and the binary threshold result.
4) logFile (argv [7]: to log the progress of the program.

****
III. Data structure:
****

- Enhancement class
    - (int) numRows
    - (int) numCols
    - (int) minVal
    - (int) maxVal
    - (int) maskRows
    - (int) maskCols
    - (int) maskMin
    - (int) maskMax
    - (int) newMin
    - (int) newMax
    - (int) thrVal // threshold value, from argv [3]
    - (int**) mirrorFmAry // a 2D array of size numRows + 2 by numCols + 2. dynamically allocate.
    - (int**) avgAry // a 2D array of size numRows + 2 by numCols + 2. dynamically allocate.
    - (int**) medianAry // a 2D array of size numRows + 2 by numCols + 2. dynamically allocate.
    - (int**) GaussAry // a 2D array of size numRows + 2 by numCols + 2. dynamically allocate.
    - (int**) thrAry // a 2D array of size numRows + 2 by numCols + 2. dynamically allocate
    - (int**) mask2DAry // a 2D Gaussian mask of size maskRows by maskCols to hold the Gaussian mask.
    - (int) mask1DAry [9] // to hold the 9 pixels of mask, for easy convolution computation.
    - (int) maskWeight // The total value of the mask, can be computed during loadMask.
    - (int) neighbor1DAry [9] //1-D array to hold a pixel[i,j]'s 3x3 neighbors, for easy computation.

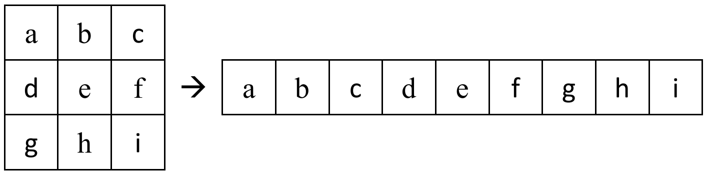
methods:

- binThreshold (inAry, outAry, thrVal, logFile) // **On your own**.

```
// if inAry[i,j] >= thrVal
        outAry[i,j] ← 1
  else
        outAry[i,j] ← 0
```

- prettyPrint (...) // **re-use code from project1**. **Don't forget to print the image header!**
- mirrorFraming (...,logFile) // **On your own**. copy row 1 to row 0 and copy row numRows+1 to row numRows,
  // then copy column 1 to column 0 and copy column numCols+1 to column numCols.

- loadImage (..., logFile) // Read from input file and load onto mirrorFmAry begin at [1][1]. **On your own**.
- (int) loadMask (..., logFile) // **On your own**.
  // load maskFile onto mask2DAry, at the same time computes and returns maskWeight .
- loadMask2Dto1D (..., logFile) // Load 3x3 pixels of mask into mask1DAry;
  // use 2 loops; do NOT write 9 assignments. **On your own**.



- loadNeighbor2Dto1D (..., logFile) // Load the 3 x 3 neighbors of mirrorFmAry (i,j) into neighbor1DAry,
  // just like the above. **On your own**.
- sort (neighborAry) // Use build-in or write your own bubble-sort method.
- computeMedian (...) // Compute 3x3 median filter on mirrorFmAry, begin at [1][1],
  // need to keep track of newMin and newMax. **See algorithm below.**
- computeAvg (..., logFile) // Compute 3x3 averaging on mirrorFmAry begin at [1][1],
  // need to keep track of newMin and newMax. Algorithm is similar to the
  // computeMedian algorithm steps. **On your own**.
- computeGauss (...) // process the mirrorFramedAry begin at [1][1]; keep track of newMin and newMax.
  // **See algorithm below**.
- (int) convolution (neighbor1DAry, mask1DAry) // **See algorithm below**.
- print1DAry (Ary, oFile) // Print from Ary[0] to Ary[8], to oFile, in one text-line. **On your own**.

***** For all "on your own methods", write the entering and leaving of methods to logFile.**

****

IV. Main(...)

****

```
Step 0: check if argc count is correct and each file can be opened.
          inFile, maskFile, and all outFiles ← open via argv []
           thrVal ← atoi (argv [3])

Step 1: numRows, numCols, minVal, maxVal ← read from inFile
         maskRows, maskCols, maskMin, maskMax ← read from maskFile
Step 2: dynamically allocate all 1-D and 2-D arrays
Step 3: maskWeight ← loadMask (maskFile, mask2DAry, logFile)
         loadMask2Dto1D (mask2DAry, mask1DAry, logFile)
Step 4: loadImage (inFile, mirrorFmAry, logFile)
Step 5: mirrorFraming (mirrorFmAry, logFile)

Step 6:  AvgFile ← "** Below is the mirror framed input image. ***"
          PrettyPrint (mirrorFmAry, AvgFile)
          computeAvg (mirrorFmAry, avgAry, logFile)
          AvgFile ← "** Below is the 3x3 averaging of the input image. ***"
          prettyPrint (avgAry, AvgFile)
          binThreshold (avgAry, thrAry, thrVal, logFile)
          AvgFile ← "** Below is the result of the binary threshold of averaging image. ***"
          prettyPrint (thrAry, AvgFile)

Step 7: MedianFile ← "** Below is the mirror framed input image. ***"
          PrettyPrint (mirrorFmAry, MedianFile)
          computeMedian (mirrorFmAry, medianAry, logFile)
          MedianFile ← "** Below is the 3x3 median filter of the input image. ***"
          prettyPrint (medianAry, MedianFile)
          binThreshold (medianAry, thrAry, thrVal, logFile)
          MedianFile ← "** Below is the result of the binary threshold of median filtered image. ***"
          prettyPrint (thrAry, MedianFile)
```

```
Step 8: GaussFile ← "** Below is the mirror framed input image. ***"
        PrettyPrint (mirrorFmAry, GaussFile)
        GaussFile ← "** Below is the mask for Gaussian filer. ***"
        PrettyPrint (mask2DAry, GaussFile)
        computeGauss (mirrorFmAry, GaussAry, logFile)
        GaussFile ← "** Below is the 3x3 Gaussian filter of the input image. ***"
        prettyPrint (GaussAry, GaussFile)
        binThreshold (medianAry, thrAry, thrVal, logFile)
        GaussFile ← "** Below is the result of the binary threshold of Gaussian filtered image. ***"
        prettyPrint (thrAry, GaussFile)
Step 9: close all files
```

****
V. computeMedian (mirrorFmAry, medianAry, logFile) // keep track of newMin and newMax
****

```
Step 0: logFile ← "** Entering computeMedian method ***"
        newMin ← 9999; newMax ← 0
Step 1: i ← 1
Step 2: j ← 1
Step 3: loadNeighbor2Dto1D (mirrorFmAry, i, j, neighbor1DAry)
       logFile ← "** Below is conversion of mirrorFmAry [i][j]  to 1D array" // write i, j values.
       print1DAry (neighobor1DAry, logFile)
Step 4: sort (neighborAry)
       logFile ← "** Below is the sorted neighbor array" // write i, j values.
       print1DAry (neighobor1DAry, logFile)
Step 5: medianAry [i,j] ← neighborAry[4]
Step 6: if newMin > medianAry [i,j]
                newMin ← medianAry [i,j]
        if newMax < medianAry [i,j]
                newMax ← medianAry [i,j]
Step 7: j++
Step 8: repeat step 3 to step 7 while j <= numCols
Step 9: i++
Step 10: repeat step 2 to step 9 while i <= numRows
Step 11: logFile ← "** Leaving computeMedian method ***"
```

****
VI. computeGauss (mirrorFmAry, GaussAry, logFile) // keep track of newMin and newMax
****

```
Step 0: logFile ← "Entering computeGauss method"
         newMin ← 9999; newMax ← 0
Step 1: i ← 1
Step 2: j ← 1
Step 3: loadNeighbor1DAry (i, j, neighbor1DAry)
       logFile ← "** Below is conversion of mirrorFmAry [i][j]  to 1D array" // write i, j values.
       PrintAry (neighobor1DAry)
       logFile ← "** Below is mask1DAry" // write i, j values.
       PrintAry (mask1DAry)
Step 4: GaussAry [i,j] ← convolution (neighbor1DAry, mask1DAry, logFile)
Step 5: if newMin > GaussAry [i,j]
                newMin ← GaussAry [i,j]
        if newMax < GaussAry [i,j]
                newMax ← GaussAry [i,j]
Step 6: j++
Step 7: repeat step 3 to step 5 while j <= numCols
Step 8: i++
Step 9: repeat step 2 to step 6 while I <= numRows
Step 10: logFile ← "Leaving computeGauss method"
```

```
*************************************************
 VII. (int) convolution (neighbor1DAry, mask1DAry, logFile)
*************************************************
Step 0: logFile ← "Entering convolution method"
        result ← 0
Step 1: i ← 0
Step 2: result += neighbor1DAry[i] * mask1DAry[i]
Step 3: i++
Step 4: repeat step 2 – step 3 while i < 9
Step 5: result ← result / maskWeight
Step 5: logFile ← "Leaving convolution method, convolution maskWeight =  ; result/maskWeight =" // write values.
Step 6: return result
```