Project 2 (C++): You are to implement the three image enhancement methods taught in class: (1) 5x5 averaging, (2) 5x5 median filter, and (3) 5x5 Gaussian filter. You may need to start coding 3 days prior to the due date.

No late submission will be accepted.

****

Project points: 12 pts

Language: C++

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

12/12 (on time): 2/19/2023 Sunday before midnight.

-12/12 (non-submission): 2/19/2023 Sunday after midnight.

*** Name your soft copy and hard copy files using the naming convention as given in the project submission requirement.

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

****

What you need to do:

1) Your will be given two input files: img1 (5 by 5) and img2 (46 by 46) and a mask file.
2) Implement your program according the specs until pass compilation.
3) Run and test your program with img1 and mask, eyeball the result of averaging, median and Gaussian in deBugFile until your program produces the correct results.
4) Run and test your program with img2 and mask
5) Include in your hardcopy *.pdf file:
    - 1 cover page (include main algorithm steps in the one cover page.)
    - Source code
    From img1 and mask
        - imgOutFile
        - AvgOutFile
        - MedianOutFile
        - GaussOutFile
        - deBugFile
    From img2 and mask
        - imgOutFile
        - AvgOutFile
        - MedianOutFile
        - GaussOutFile
        - deBugFile

****

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
(See mask1 and mask2)

c) a threshold value (argv[3]) // USE 36

****

II. Output files:

1) AvgOutFile (argv[4]): prettyPrint the threshold result of 5x5 averaging.
2) MedianOutFile (argv[5]): prettyPrint the threshold result of 5x5 median filter.
3) GaussOutFile (argv[6]): prettyPrint the threshold result of 5x5 Gaussian filter.
4) deBugFile (argv[7]): Print mirrorFramedAry, avgAry, medianAry, and GaussAry after reformatting.

****

III. Data structure:

****

```
 - Enhancement class
        - (int) numRows
        - (int) numCols
        - (int) minVal
        - (int) maxVal
        - (int) maskRows
        - (int) maskCols
        - (int) maskMin
        - (int) maskMax
        - (int) thrVal  // from argv[3]
        - (int) mirrorFramedAry[][] // a 2D array of size numRows + 4 by numCols + 4. dynamically allocate.
        - (int) avgAry [][] // a 2D array of size numRows + 4 by numCols + 4. dynamically allocate.
        - (int) medianAry [][] // a 2D array of size numRows + 4 by numCols + 4. dynamically allocate.
        - (int) GaussAry [][] // a 2D array of size numRows + 4 by numCols + 4. dynamically allocate.
        - (int) thrAry // a 2D array of size numRows + 4 by numCols + 4. dynamically allocate.
                                  // to hold the threshold result for each operation.
        - (int) neighborAry [25] //1-D array to hold a pixel[i,j]'s 5x5 neighbors to ease computation.
        - (int) maskAry [25] // to hold the 25 pixels of mask to ease computation.
        - (int) maskWeight // The total value of the mask, to be computed in loadMask1DAry method.

         methods:
        - binaryThreshold (inAry, thrAry) // Reuse code from your project01.
        - loadImage (...) // On your own.  Read from inFile and load onto mirrorFramedAry begin at [2][2].
        - mirrorFraming (...) // On your own. The algorithm of Mirror framing was taught in class
        - loadMaskAry (...) // On your own. Load 25 pixels of mask into maskAry;
                         //  use loops; do NOT write 25 assignments.
        - loadNeighborAry (...) // On your own. Load the 5 x 5 neighbors of mirrorFramedAry (i,j) into neighborAry,
                            // using loops;  do NOT write 25 assignments.
        - sort (neighborAry) // Use any build-in sorting algorithm or write your own sorting method.
        - computeMedian (...) // process the mirrorFramedAry begin at [2][2]. See algorithm below.
        - computeAvg (...) // process the mirrorFramedAry begin at [2][2]. On your own.
        - computeGauss (...) // process the mirrorFramedAry begin at [2][2]. See algorithm below.
        - (int) convolution (neighbor1DAry, mask1DAry) // See algorithm below.
        - imgReformat (...) // see algorithm below.
        - prettyPrint (Ary, outFile) // For nice visual, use font – "Courier New"  to line-up pixels
                                  if Ary(i, j ) > 0
                                        outFile ← Ary[i][j] follow by one blank space
                                  else
                                        outFile ← two blank spaces
```

****

IV. Main(...)

****

```
Step 0: open inFile, maskFile via argv[]
        open imgOutFile, AvgOutFile, MedianOutFile, GaussOutFile via argv[]
        thrVal ← get from 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: loadMaskAry (maskFile, maskAry)
Step 4: loadImage (inFile, mirrorFramedAry)
Step 5: mirrorFraming (mirrorFramedAry)
Step 6: imgReformat (mirrorFramedAry, deBugFile)
Step 7:  computeAvg (mirrorFramedAry, avgAry)
        imgReformat (avgAry, deBugFile)
        binaryThreshold (avgAry, thrAry)
        prettyPrint (thrAry, AvgOutFile)

Step 8: computeMedian (mirrorFramedAry, medianAry)
        imgReformat (medianAry, deBugFile)
        binaryThreshold (medianAry, thrAry)
        prettyPrint (thrAry, MedianOutFile)

Step 9: computeGauss (mirrorFramedAry, GaussAry)
        imgReformat (GaussAry, deBugFile)
        binaryThreshold (GaussAry, thrAry)
        prettyPrint (thrAry, GaussOutFile)

Step 10: close all files
```

****

V. computeMedian (mirrorFramedAry, medianAry)

****

```
step 1: i ← 2
step 2: j ← 2
step 3: loadNeighborAry (mirrorFramedAry, i, j, neighborAry)
step 4: sort (neighborAry)
step 5: medianAry [i, j] ← neighborAry [12]
step 6: j++
step 7: repeat step 3 to step 7 while j < (numCols + 2)
step 8: i++
step 9: repeat step 2 to step 9 while i < (numRows + 2)
```

****

VI. computeGauss (mirrorFramedAry, GaussAry) // keep track of newMin and newMax

****

```
step 1: i ← 2
step 2: j ← 2
step 3: loadNeighbor1DAry (mirrorFramedAry, i, j, neighborAry)
step 4: GaussAry [i, j] ← convolution (neighborAry, maskAry)
step 5: j++
step 6: repeat step 3 to step 5 while j < (numCols + 2)
step 7: i++
step 8: repeat step 2 to step 6 while i <  (numRows + 2)
```

****

VII. (int) convolution (neighborAry, maskAry)

****

```
step 0: result ← 0
step 1: i ← 0
step 2: result += neighborAry[i] * maskAry[i]
step 3: i++
step 4: repeat step 2 – step 3 while i < 25
step 5: return (result /maskWeight)
```

```
*******************************
VIII. imgReformat (inAry, outFile)
*******************************
Step 1: outFile← output numRows, numCols, minVal, maxVal
Step 2: str ← to_string (maxVal) // C++ build-in
          Width ← length of str
Step 3: r ← 2
Step 4: c ← 2
Step 5: outFile ← inAry[r][c]
Step 6: str ← to_string (inAry[r][c])
          WW ← length of str
Step 7:  outFile ← one blank space
          WW ++
Step 8: repeat step 7 while WW < Width
Step 9: c++
Step 10: repeat Step 5 to Step 9 while c < (numCols + 2)
Step 11: r++
Step 12: repeat Step 4 to Step 10 while r < (numRows + 2)
```