Project 2 (in C++): You are to implement the three image enhancement methods taught in class: (1) 3X3 averaging, (2) 3x3 median filter,(3)5x5 corner preserving filter.

****
Add the following into your #include:

```
#include<string>
using namespace std;
```

****
Hard copy includes:

- Cover page
- Source code
- rfImg file
- AvgOutImg file
- AvgThrImg file
- AvgPrettyPrint file
- MedianOutImg file
- MedianThrImg file
- MedianPrettyPrint file
- CPOutImg file
- CPThrImg file
- CPrettyPrint file

** You must use a fix font -- "courier new", and choose a font size so that When printing an image file, it will fit in one page.

****
Language: (C++)
****
Project points: 10 pts
Due Date: Soft copy (*.zip) and hard copies (*.pdf):

-0 2/21/2021 Sunday before midnight
-1 for 1 day late: 2/22/2021 Monday before midnight
-2 for 2 days late: 2/22/2021 Tuesday before midnight
-10/10: 2/22/2021 Tuesday after midnight

*** Name your soft copy and hard copy files using the naming convention as given in the project submission requirement discussed in a lecture and is posted in Google Classroom.

*** 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.

****
I. Input files:

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

b) **threshold value (argv[2]) // USE 30**

```
*******************************
II.   Output files:
      1) rfImg (argv[3]): the input image after reformatting.
      2) AvgOutImg(argv[4]): The image of the result of 3x3 average filter,
                             after reformatting.
      3) AvgThrImg(argv[5]): The threshold result of 3x3 average filter,
                             after reformatting.
      4) AvgPrettyPrint(argv[6]): The pretty print of the threshold result of
                             average filter.
      5) MedianOutImg(argv[7]): The image of the result of 3x3 median filter,
                             after reformatting.
      6) MedianThrImg(argv[8]): The threshold result of 3x3 median filter,
                             after reformatting.
      7) MedianPrettyPrint(argv[9]): The pretty print of the threshold result
                             of median filter.
      8) CPOutImg(argv[10]): The image of the result of 5x5 corner preserve
                             filter, after reformatting.
      9) CPThrImg(args[11]): The threshold result of 5x5 corner preserve
                             filter, after reformatting.
      10) CPPrettyPrint(args[12]): The pretty print of the threshold result
                             of corner preserve filter.
*******************************
III. Data structure:
*******************************
 - imageProcessing class
      - (int) numRows
      - (int) numCols
      - (int) minVal
      - (int) maxVal
      - (int) newMin
      - (int) newMax
      - (int) thrVal // from argv[2]
      - (int) neighborAry [9] // You may consider using this
              // 1-D array to hold the 3x3 neighbors of a pixel
              // during the computation of average and median operations.
      - (int) CPmasks[8][5][5] // This 3D array is used in the corner
              //preserving averaging. These 8 masks of 5x5 are designed to
              //compute the averages of the 8 groups in a pixel's 5x5
              //neighborhood without having to indexing the coordinates of 9
              //pixels in each of the 8 groups. The 8 masks are posted in Google
              //classroom.
              // The detail of the usage of these masks is given in the lecture.
      - (int) neighbor5x5[5][5] // for store the 5x5 neighbors of a pixel.
      - (int **) mirror3by3Ary // a 2D array, dynamically allocate
                 //at run time of size numRows + 2 by numCols + 2.
                 // This array is for loading the input image into
                 // the inside the frame of the array and to be use by
                 // 3x3 averaging (mean filter) and median filter.
      - (int **) mirror5by5Ary // a 2D array, dynamically allocate
                 //at run time of size numRows + 4 by numCols + 4.
                 // This array is for loading the input image into
                 // the inside the frame of the array and to be use by
                 // 5x5 corner preserve averaging.
```

```
- (int **) avgAry // a 2D array, dynamically allocate at run time
                  // of size numRows + 2 by numCols + 2 for storing
                  // the result of the 3x3 averaging
- (int **)medianAry // a 2D array, dynamically allocate at run time
                  // of size numRows + 2 by numCols + 2 for storing
                  // the result of the 3x3 median filter.
- (int **) CPAry // a 2D array, dynamically allocate at run time
                  // of size numRows + 4 by numCols + 4 for storing
                  // the result of the 5x5 corner preserving averaging
```

methods:

```
- threshold (...)// see algorithm below.
- imgReformat (...) // see algorithm below.
- loadCPmasks (...) // Either hard coded or read from files
- loadneighbors (...) // load the 5x5 neighbors of a pixel.
- loadImage (...)// On your own!!
      // read from input file and load the image onto mirror3by3Ary and
      // mirror5by5Ary.
- mirrorFraming (Ary, frameSize) // On your own!! See lecture note.
          // The method should be able handle 3x3 mirrorframe and 5x5
          // mirrorframe. For 3x3, frame size is 1, for 5x5, frame size
          // is 2.
- ComputeAvg (...) // see algorithm below.
      // Scans thru all pixels inside the frame of the mirror3by3Ary,
      // apply avg3x3 on each pixel, then, outputs the result to avgAry;
      // it also keeps track of newMin and newMax during the process.
- computeMedian(...) // see algorithm below.
      // Scans thru all pixels inside the frame of the mirror3by3Ary,
      // apply median3x3 on each pixel, then, outputs the result to
      // medianAry; it also keeps track of newMin and newMax during the
      // process.
- computeCPfilter (...) // see algorithm below.
      // Scans thru all pixels inside the frame of the mirror5by5Ary,
      // applying the corner preserving algorithm, then, outputs the
      // result to CPAry; it also keeps track of newMin and newMax
      // during the process.
- sort (neighborAry) // Used by median filter method.  You may use any
      //sorting algorithm.  On your own!!
- (int) avg3x3 (i, j) // On your own!! See lecture note.
      // computes and returns the average of the pixel (i, j)'s 3x3
      // neighborhood.
- (int) median3x3 (i, j) //On your own!! See lecture note.
      // computes and returns the median value of the pixel (i, j)'s 3x3
      // neighborhood.
- (int) CP5x5 (i, j) //On your own!! See lecture note.
      // The method loads the 5x5 neighbors of mirror5by5Ary[i, j] onto
      // CPneighbor5x5 array, then apply convolution using CPmasks to
      //get the averages; then computes the differences between
      //mirror5by5Ary[i, j] and each of the 8 averages, then returns the
      //average of a group having the minimum differences.
```

```
-(int) convolution (...) // On your own!! See lecture note.
       // Compute the convolution using a given 5x5 mask
       // onto the loaded Pneighbor5x5 and returns the result.
 - AryToFile (ary, outFile, frameSize)
       // on your own!
       // It prints the image header: numRows numCols newMin newMax
       //to outFile, then, prints the ary without the frames.
 - prettyPrint (inAry, outFile) // print without the frames.
          // if inAry [i][j] > 0
                outFile ← inAry [i][j] follows by one blank space
          else
                outFile ← "." follows by one blank space
```

```
*******************************
IV. main(...)
*******************************
step 0: open inFile  and open all outfiles
        thrVal ← get from argv[2]
step 1: numRows, numCols, minVal, maxVal ← read from inFile
        newMin ← minVal
        newMax ← maxVal

step 2: loadImage (inFile)

step 3: mirrorFraming (mirror3by3Ary,1)
        imgReformat (mirror3by3Ary, rfImg)
step 4: ComputeAvg(...)
        imgReformat (avgAry, AvgOutImg, 1)
        threshold (avgAry, thrAry, 1)
        AryToFile (thrAry, AvgThrImg, 1)
        prettyPrint (thrAry, AvgPrettyPrint, 1)
step 5: computeMedian (...)
        imgReformat (medianAry, MedianOutImg, 1)
        threshold (medianAry, thrAry, 1)
        AryToFile (thrAry, MedianThrImg, 1)
        prettyPrint (thrAry, MedianPrettyPrint, 1)
Step 6: mirrorFraming (mirror5by5Ary, 2)
Step 7: computeCPfilter (...)
        imgReformat (CPAry, CPOutImg, 2)
        threshold (CPAry, thrAry, 2)
        AryToFile (thrAry, CPThrImg, 2)
        prettyPrint (thrAry, CPPrettyPrint, 2)
step 8: close all files
```

```
************************************
V. computeAvg (...)
************************************
     // process the entire ary, keep track of newMin and newMax
step 0: newMin ← 9999; newMax ← 0
step 1: r ← 1
step 2: c ← 1
step 3: avgAry [r,c] ← avg3x3 (r, c)
step 4: if newMin > avgAry [r,c]
           newMin ← avgAry [r,c]
        if newMax < avgAry [r,c]
           newMax ← avgAry [r,j]
step 5: c++
step 6: repeat step 3 to step 5 while c < numCols+1
step 7: r++
step 8: repeat step 2 to step 7 while r < numRows+1
************************************
VI. computeMedian (...)
************************************
     // process the entire ary, keep track of newMin and newMax
step 0: newMin ← 9999; newMax ← 0
step 1: r ← 1
step 2: c ← 1
step 3: medianAry [r,c] ← median3x3 (r, c)
step 4: if newMin > medianAry [r,c]
           newMin ← medianAry [r,c]
        if newMax < medianAry [r,c]
           newMax ← medianAry [r,j]
step 5: c++
step 6: repeat step 3 to step 5 while c < numCols+1
step 7: r++
step 8: repeat step 2 to step 7 while r < numRows+1
************************************
VI. computeCPfilter (...)
************************************
     // process the entire ary, keep track of newMin and newMax
step 0: newMin ← 9999; newMax ← 0
step 1: r ← 2
step 2: c ← 2
step 3: CPAry [r,c] ← CP5x5 (r, c)

step 4: if newMin > CPAry [r,c]
           newMin ← CPAry [r,c]
        if newMax < CPAry [r,c]
           newMax ← CPAry [r,j]
step 5: c++
step 6: repeat step 3 to step 5 while c < numCols+2

step 7: r++
step 8: repeat step 2 to step 7 while r < numRows+2
```

```
*******************************
VII. imgReformat (inAry, OutImg, frameSize)
*******************************
Step 1: OutImg ← output numRows, numCols, newMin, newMax
Step 2: str ← to_string(newMax)  // a method in C++ string class
        Width ← length of str
Step 3: r ← frameSize
Step 4: c ← frameSize
Step 5: OutImg ← inAry[r][c]
Step 6: str ← to_string (inAry[r][c])
        WW ← length of str

Step 7: OutImg ← 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 + frameSize)
Step 11: r++
Step 12: repeat Step 4 to Step 10 while c < (numCols + frameSize)

*******************************
VII. threshold (ary1, ary2, frameSize)
*******************************
step 0: newMin ← 0
        newMax ← 1
step 1: r ← frameSize
step 2: c ← frameSize
step 3: if ary1[r][c] >= thrVal
           ary2[r][c] ← 1
        else
           ary2[r][c] ← 0
step 4: c++
step 5: repeat step 3 to step 4 while c < (numCols + frameSize)
step 6: r++
step 7: repeat step 2 to step 6 while r < (numRows + frameSize)
```