Project 4 (C++) : Implementation of the four basic Morphology Operations.

*** What do you need to do:
Step 0:

- Implement your program according to the specs given below.
- you will be provided with three data (data1, data2, data3) and three structuring elements (elm1, elm2 and elm3) and the answers for you to verify the correctness of your program.
- run your program using data1 and elm1 on all four morphological operations then compare the result with the answers.
- run your program using data1 and elm2 on all four morphological operations then compare the result with the answer.
- run your program using data2 and elm2 on all four morphological operations then compare your result with the answer.
- run your program using data3 and elm3 on all four morphological operations then compare your result with the answer.
- continue to debug your program until the result of your program produce the results as shown in answers. **You will receive 6 pts out of 12 if you accomplished this.**
  // include the results (16 in total) in your pdf hard copies,

** You will be given two data (img1 and img2) to do the following:
Step 1: For img1:

- img1 contains some large blob-shape objects and some small blob-shape objects and some random noises.
- you are to design a structuring element (blobElm)– based on your observation of size of the larger blob-shape objects. The goal is to extract only those larger blob-shape objects but without the smaller circular object nor those random noise.
- run your program using img1 with blobElm you designed on each of all four 4 morphological operations (4 results in total) and modify your blobElm until the result of your program extract only those large blob-shape objects. **You will receive 3 pts out of 12 if you accomplished this.**
  // include the 4 results in your pdf hard copies and indicate which of operation yield the optimal result.

Step 2: For img2:

- img2 contains lines in vertical, horizontal, right diagonal (/) and left diagonal (\) with random noise.
- you are to design four structuring elements (vertElm, horiElm, rightElm and leftElm), to extract lines but without random noise.
- run your program using img2 with vertElm to produce 4 results.
- run your program using img2 with horiElm to produce 4 results.
- run your program using img2 with rightElm to produce 4 results.
- run your program using img2 with leftElm to produce 4 results.
- modify your structuring elements until your program extract all lines. **You will receive 3 pts out of 12 if you accomplished this.**

Your hard copies include:
- cover sheet
- source code
- print all results produced from Step 0
- print all results produced from Step 1
- print all results produced from Step 2

****
Language: C++
Project points: 10 pts
Due Date: Soft copy (*.zip) and hard copies (*.pdf):
+1 (11/10 pts): early submission, 10/15/2022, Saturday before midnight
-0 (10/10 pts): on time, 10/19/2022, Wednesday before midnight
-1 (9/10 pts): 1 day late, 10/20/2022, Thursday before midnight
-2 (8/10 pts): 2 days late, 10/21/2022, Friday before midnight
(-10/10 pts): non submission, 10/21/2022, Friday 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. Inputs: There are two input files.

****

a) Input1 (argv[1]): a txt file representing a binary image with header.

b) Input2 (argv[2]): a txt file representing a binary image of a structuring element with header and the origin of the structuring element. The format of the structuring element is as follows: 1th text line is the header; the 2nd text line is the position (w.r.t. index) of the origin of the structuring element then follows by the rows and column of the structuring element.
For example:

5 5 0 1 // 5 rows, 5 columns, min is 0, max is 1: 2-D structuring element
2 2 // origin is at row index 2 and column index 2.
0 0 1 0 0
0 0 1 0 0
1 1 **1** 1 1
0 0 1 0 0
0 0 1 0 0

** Note: when a structure element contains zeros, only those 1's to be used in the matching in the erosion!

Another example:

3 3 1 1 // 3 rows, 3 columns, min is 1, max is 1: 2-D structuring element
1 1 // origin is at row index 1 and column index 1.
1 1 1
1 **1** 1
1 1 1

Another example:

1 5 1 1 // 1 rows, 5 columns, min is 1, max is 1: 1-D structuring element
0 2 // origin is at row index 0 and column index 2.
1 1 **1** 1 1

****

II. Outputs: (All of the following output files need to be included in your hard copies!)

- dilateOutFile (argv[3]): the result of dilation image with header, the same dimension as imgFile
- erodeOutFile (args[4]): the result of erosion image with header, the same dimension as imgFile
- closingOutFile (args[5]): the result of closing image with header, the same dimension as imgFile
- openingOutFile (args[6]): the result of opening image with header, the same dimension as imgFile
- prettyPrintFile (args[7]): pretty print which are stated in the algorithm steps

*** Note: When you run your program, please name your output files as given in the above.
*** NO HARD coded file names in the program**, you will receive the score of 0/12,** if you hard code file name in this project!!!

****

III. Data structure:

****

- a Morphology class
  - (int) numImgRows
  - (int) numImgCols
  - (int) imgMin
  - (int) imgMax
  - (int) numStructRows
  - (int) numStructCols
  - (int) structMin
  - (int) structMax
  - (int) rowOrigin
  - (int) colOrigin

- (int) rowFrameSize // set to (numStructRows / 2), integer division, i.e., 3/2 is 1; 4/2 is 2; 5/2 is 2.
- (int) colFrameSize // set to (numStructCols / 2).
- (int) extraRows // set to (rowFrameSize * 2)
- (int) extraCols // set to (colFrameSize * 2)
- (int) rowSize // set to (numImgRows + extraRows)
- (int) colSize // set to (numImgCols + extraCols

- (int**) zeroFramedAry // a dynamically allocate 2D array, size of rowSize by colSize, for the input image.
- (int**) morphAry // Same size as zeroFramedAry.
- (int **) tempAry // Same size as zeroFramedAry.
  // tempAry is to be used as the intermediate result in opening and closing operations.
- (int **) structAry //a dynamically allocate 2D array of size numStructRows by numStructCols, for structuring element.

Methods:

- zero2DAry (Ary, nRows, nCols) // Set the entire Ary (nRows by nCols) to zero.
- loadImg (…) // load imgFile to zeroFramedAry inside of frame, begins at (rowOrigin, colOrigin). On your own!
- loadstruct (…) // load structFile to structAry. On your own!
- ComputeDilation (inAry, outAry) // process every pixel in inAry, put result to outAry // see algorithm below.
- ComputeErosion (inAry, outAry) // process every pixel in inAry, put result to outAry // see algorithm below.
- ComputeOpening (inAry, outAry, tmp) // see algorithm below.
- ComputeClosing (inAry, outAry, tmp) // see algorithm below.
- onePixelDilation (i, j, inAry, outAry) // Perform dilation on pixel (i, j) with structAry. // See algorithm below.
- onePixelErosion (i, j, inAry, outAry) // Perform erosion on pixel (i, j) with structAry. // See algorithm below.
- AryToFile (Ary, outFile) // output the image header (from input image header)
  //then output the rows and cols of Ary to outFile *excluding* the framed borders of Ary.
- prettyPrint (Ary, outFile) // Remark: use "Courier new" font and small font size to fit in the page.
  // if Ary [i, j] == 0 output ". " // a period follows by a blank
  // else output "1 " // 1 follows by a blank

****

IV. Main(...)

****

```
step 0: imgFile, structFile, dilateOutFile, erodeOutFile, openingOutFile, closingOutFile, prettyPrintFile ← open

step 1:  numImgRows, numImgCols, imgMin, imgMax ← read from imgFile
         numStructRows, numStructCols, structMin, structMax ← read from structFile
         rowOrigin, colOrigin ← read from strucFile

step 2:  zeroFramedAry, structAry, morphAry, tempAry ← dynamically allocate // see description in the above

step 3: zero2DAry(zeroFramedAry, rowSize, colSize) // see description in the above

step 4: loadImg (imgFile, zeroFramedAry) // see description in the above
        prettyPrint (zeroFramedAry, prettyPrintFile) // write a meaningful caption before prettyPrint

step 5:  zero2DAry(structAry, numStructRows, numStructCols)
         loadstruct (structFile, structAry) // see description in the above
         prettyPrint (structAry, prettyPrintFile) // see description in the above

step 6:  zero2DAry(morphAry, rowSize, colSize)
         ComputeDilation (zeroFramedAry, morphAry) // see algorithm below
         AryToFile (morphAry, dilateOutFile) // see description in the above
         prettyPrint (morphAry, prettyPrintFile) // write a meaningful caption before prettyPrint

step 7:  zero2DAry(morphAry, rowSize, colSize)
         ComputeErosion (zeroFramedAry, morphAry) // see algorithm below
         AryToFile (morphAry, erodeOutFile)
         prettyPrint (morphAry, prettyPrintFile) // write a meaningful caption before prettyPrint

step 8:  zero2DAry(morphAry, rowSize, colSize)
         ComputeOpening (zeroFramedAry, morphAry, tempAry) // see algorithm below
         AryToFile (morphAry, openingOutFile)
         prettyPrint (morphAry, prettyPrintFile) // write a meaningful caption before prettyPrint

step 9:  zero2DAry(morphAry, rowSize, colSize)
         ComputeClosing (zeroFramedAry, morphAry, tempAry) // see algorithm below
         AryToFile (morphAry, closingOutFile)
         prettyPrint (morphAry, prettyPrintFile) // write a meaningful caption before prettyPrint

step 10: close all files
```

****

V. ComputeDilation (inAry, outAry)

****

```
         // process dilation on each pixel inside of zeroFramedAry
step 1: i ← rowFrameSize
step 2: j ← colFrameSize
step 3: if inAry [i,j] > 0
          onePixelDilation (i, j, inAry, outAry)  // only processing one pixel inAry[i,j]
step 4: j++
step 5: repeat step 3 to step 4 while j < (colSize)
step 6: i++
step 7: repeat step 2 to step 6 while i < (rowSize)
```

****

VI. ComputeErosion (inAry, outAry) // process dilation on each pixel in the entire zeroFramedAry

****

```
step 1: i ← rowFrameSize
step 2: j ← colFrameSize
step 3: if inAry[i,j] > 0
          onePixelErosion (i, j, inAry, outAry) // only processing one pixel inAry[i,j]
step 4: j++
step 5: repeat step 3 to step 4 while j < (colSize)
step 6: i++
step 7: repeat step 2 to step 6 while i < (rowSize)
```

****

VI. onePixelDilation (i, j, inAry, outAry)

****

```
step 0 : iOffset ← i - rowOrigin
         jOffset ← j - colOrigin
           // translation of image's coordinate (i, j) with respected to the origin of the structuring element
step 1: rIndex ← 0
step 2: cIndex ← 0
step 3: if (structAry[rIndex][cIndex]  > 0)
          outAry[iOffset + rIndex][jOffset + cIndex] ← 1
step 4: cIndex ++
step 5: repeat step 3 to step 4 while cIndex <  numStructCols
step 6: rIndex ++
step 7: repeat step 2 to step 6 while rIndex < numStructRows
```

****

VII. onePixelErosion (i, j, inAry, outAry)

****

```
step 0 : iOffset ← i - rowOrigin
         jOffset ← j - colOrigin
           // translation of image's coordinate (i, j) with respected of the origin of the structuring element
         matchFlag ← true
step 1: rIndex ← 0
step 2: cIndex ← 0
step 3: if (structAry[rIndex][cIndex] > 0) and (inAry[iOffset + rIndex][jOffset + cIndex] ) <= 0)
            matchFlag ← false
step 4: cIndex ++
step 5: repeat step 3 to step 4 while (matchFlag == true) and (cIndex <  numStructCols )
step 6: rIndex ++
step 7: repeat step 2 to step 6 while (matchFlag == true) and (rIndex < numStructRows)
step 8: if matchFlag == true
                outAry[i][j] ← 1
         else
                outAry[i][j] ← 0
```

****

VIII. ComputeClosing (zeroFramedAry, morphAry, tempAry)

****

```
step 1: ComputeDilation (zeroFramedAry, tempAry)
step 2: ComputeErosion (tempAry, morphAry)
```

****

IV. ComputeOpening (zeroFramedAry, morphAry, tempAry)

****

```
step 1: Compute Erosion (zeroFramedAry, tempAry)
step 2: ComputeDilation (tempAry, morphAry)
```