Project 3 (C++): Implementation of the four basic Morphology Operations.

- Implement your project using the specs below.
- You will have two image files and four structuring elements to test your program.
- Run your program 4 times:
  a) test1: imgFile1 with elm1
  b) test2: imgFile1 with elm2
  c) test3: imgFile2 with elm3
  d) test4: imgFile2 with elm4

Your hard copies include:
- cover sheet
- program source code
- print all output files of test1
- print all output files of test2
- print all output files of test3
- print all output files of test4

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

-0 (10/10) 9/24/2021 Friday before midnight
+1 (11/10) 9/20/2021 Monday before midnight
-1 (9/10) for 1 day late: 9/25/2021 Saturday before midnight
-2 (8/10) for 2 days late: 9/26/2021 Sunday before midnight
(-10/10) non submission: 9/26/2021 Sunday 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:
****

a) imgFile (argv[1]): a txt file representing a binary image with header.
b) structFile (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:
1st 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 dilation and 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, without framed boarders.
- erodeOutFile (argv [4]): the result of erosion image with header, the same dimension as imgFile
- closingOutFile (argv [5]): the result of closing image with header, the same dimension as imgFile
- openingOutFile (argv [6]): the result of opening image with header, the same dimension as imgFile
- prettyPrintFile (argv [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, -2 points 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. // On your own!
- 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 Ary [i, j] 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)
*******************************
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)
```