Project 8 (C++) : K-Curvature edge detector as taught in lecture and in lecture notes. A very easy project.

You will be given four data files: dataA, dataB, dataC, and dataD, run your program four times for each data

Include in your hard copy:
- cover page
- source code
- outFile1, outFile2, outFile3 for dataA
- outFile1, outFile2, outFile3 for dataB
- outFile1, outFile2, outFile3 for dataC
- outFile1, outFile2, outFile3 for dataD

****

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

10 on time: 5/11/2021 Tuesday before midnight
-1 for 1 day late: 5/12/2021 Wednesday before midnight
-2 for 2 days late: 5/13/2021 Thursday before midnight
-10/10: 5/13/2021 Thursday after midnight
- 5/10: does not pass compilation
0/10: program produces no output
0/10: did not submit hard copy.

*** Follow "Project Submission Requirement" to submit your project.

****

I. Inputs: There are two inputs.

a) inFile (argv[1]) : A text file contains a list of boundary points of an object in an image.
The format of the input is as follows:

```
#rows #cols minVal maxVal // image header
label     // the label of the object
r1 c1
r2 c2
r3 c3
:
```

b) K (from console): ask the user for K to be used in the K-curvature computation.

****

II. Outputs: There are three output files.

a) outFile1 (argv[2]): The result of the K-curvature of the object boundary points.
The format of this output file is as follows:

```
#rows #cols minVal maxVal // image header
label     // the label of the object.
#pts //  the number of boundary points
r1 c1 1 // not a corner
r2 c2 9 // a corner (use 9 for corner indicator for the K-curvature)
r3 c3 1 // not a corner
:
:
```

b) outFile2 (argv[3]): Pretty print (displaying)of the result of the K-curvature corner detection, as in an image, where corner points are printed as 8 and non-corner points are printed as 1.

c) outFile3 (argv[4]): for all debugging output

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

- An image class
  - friend of kCurvature class
    - numRows (int)
    - numCols (int)
    - minVal (int)
    - maxVal (int)
    - imgAry (int**) // a 2D array for display, initially set to 0
  - methods:
    - constructor(…)
    - plotPt2Img (…) // put each point (x, y)'s corner value (1 or 9) at ImgAry(x, y)
    - reformatPrettyPrint (…) // reuse code from your previous project
- A boundaryPt class
  - friend of kCurvature class
    - (int) x
    - (int) y
    - (double) curvature
    - (int) localMax
    - (int) corner // 1 means it is not a corner or 9 means it is a corner
- A kCurvature class
    - (int) K // Ask the user from console
    - (int) numPts // # of boundary pts
    - Q (int) // an index of the array, initially set to 0
    - P (int) // an index of the array, initially set to K
    - R (int) // an index of the array, initially set to 2*K
    - (boundaryPt *) PtAry // an 1D array of boundaryPt class size is numPts,
      // need to dynamically allocate.
      // use mod function to compute the curvature for the beginning of
      // the K points without extending the tail of the array,
  - methods:
  - constructors (…)
  - initialization (…) // See algorithm below
  - cornerDetection (…) // See algorithm below
  - (int) countPts (…) // reads and returns the count of the boundary points in the input file.
  - storePt (x, y, index) // the x, y to PtAry[index]
  - computeCurvature (Q,P,R) // taught in class
  - computeLocalMaxima (PtAry)
    // P(i) is a local maxima if in its 1 X 5 neighborhood if the curvature
    // of p(i) is >= the curvatures of
    // its linear neighbors: p(i-2), p(i-1), p(i+1), p(i+2)
    // returns 1 if yes, returns 0 if not.
  - (int) markCorner (PtAry) // go thru the entire PtAry, i = 0 to numPts -1
    // set PtAry[i]-> corner to 9
      if a) PtAry [i] is a local maxima &&
        b) in its 1X5 neighborhood, only PtAry [i-1] or PtAry [i+1] can be a local maxima
    // otherwise, set PtAry[i]-> corner to 1

  - printBoundary(outFile1)
    // output only (x, y, corner) of the entire PtAry to outFile1 in format given in the above.
  - display(…) // plot PtAry to imgAry
  - printPtAry(outFile3) // For debugging, print the content of the entire PtAry to outFile3

```
*******************************
IV.  main ()
*******************************
        // Algorithm may contain bugs, debugging is yours and report bugs to Dr. Phillips.

Step 0: inFile, outFile1, outFile2, outFile3 ← open input files
        numRows, numCols, minVal, maxVal ← get from inFile
        label ← get from inFile
        K ← ask the user from console
        imgAry ← dynamically allow
Step 1: initialization (inFile)
        printPtAry (outFile3)
Step 2: cornerDetection (…)
        printPtAry(outFile3)
Step 3: computeLocalMaxima (PtAry) for all point in PtAry[index], index from 0 to numPts-1
Step 4: markCorner (PtAry)
Step 5: printBoundary(outFile1)
Step 6: plotPt2Img() // put each point (x, y)'s corner value (1 or 9) at Img(x, y)
Step 7: reformatPrettyPrint (imgAry, outFile2) // output imgAry to outFile2
Step  8: close all files

*******************************
V.  initialization (inFile)
*******************************
        // Algorithm may contain bugs, debugging is yours and report bugs to Dr. Phillips.

Step 1: numPts ← countPts (inFile)
Step 2:  close inFile
        inFile ← open the input file the second time.
Step 3:  index  ←  0
Step 4: (x, y) ← read from inFile
Step 5:  storePt (x, y, index) // store x, y to PtAry[index]
Step 6 : index ++;
Step 7 : Repeat step 4 - step 6 until inFile is empty

*******************************
VI. cornerDetection (…)
*******************************
        // Algorithm may contain bugs, debugging is yours and report bugs to Dr. Phillips.

Step 0:  Q ← 0
        P ← K
        R ← 2* K
Step 1: index ← P
Step 2: curvature ← computeCurvature (Q, P, R)
Step 3: PtAry[index]->curvature ←  curvature
Step 4: outFile3 ← print Q, P, R, index, x, y, curvature of PtAry[index] // for debugging to see the curvature.
Step 5: Increment Q, P, R by 1 // each need to mod with numPts
Step 6: repeat step 2 to step  until P == K-1
```