Project 7 (C++): Chain code for image compression (lossless compression) and object recognition (via) boundary Pattern Analysis. You may assume the objects in the image do not have holes in them.

*** Given a labelled file and a property file, what your Chain code program will do as follows:

1) Opens the labelled file and the property file
2) Allocate an imageAry with extra 2 rows and extra 2 cols; initialized the array to zero.
3) Read and load the labelled file onto the imageAry begins at (1,1).
4) Create a chainCode file for output. The name of the chainCode file is to be created during the run time of your program, using the original file name with an extension "_chainCode.txt". For example, if the name of the input file is "img1", then the name of the compressed file should be "img1_chainCode.txt".
5) Read the image header and write the header to chainCode file.
6) Get each CC property from property file
7) For each CC, for easy programming, we allocate a CCAry, the same size as imageAry, then
    a) load all pixels with the same label as CC.label in imageAry to CCAry
    b) apply chain-code algorithm on CCAry to produce chain code.
8) Close the chainCode file (img1_chainCode.txt)
9) Re-open the chainCode file.
10) Using chainCode file to construct the boundary of all objects in the labelled file and store in boundaryAry.
11) Output boundaryAry (inside the frame) to boundary file.
12) The name of the boundary file is to be created during the run time of your program, using the original file name with an extension "_Boundary.txt". For example, if the name of the input file is "img1", then the name of the compressed file should be "img1_ Boundary.txt".

You will be given two sets of files: a) img1CC and img1Property, and b) img2CC and img2Property.

a) Run your chainCode program using img1CC and img1Property to get chain code for each object. (the format for chain code output is given below.)
b) Run your chainCode program using img2CC and img2Property do the same as the above.
c) Print Img1CC on a piece of paper. Hand-trace the boundary of Img1CC, similar the illustration in the lecture note.
d) Include in your hard copy:
    - Cover page
    - The hand tracing of Img1CC and write the traced chain-code.
        (Check to see if your hand traced chain code is the same as your program produces.
    - Source Code
    - Print img1CC and img1Property
    - Print ChainCodeFile for Img1CC
    - Print Boundary file for img1CC
    - Print img2CC and img2Property
    - Print ChainCodeFile for Img2CC
    - Print Boundary file for img1CC

****

Language: C++
Points: 12 pts
Due Date: <u>Soft copy (*.zip) and hard copies (*.pdf)</u>:

-0 (12/12 pts): on time, 4/24/2022 Sunday before midnight
+1 (13/12 pts): early submission, 4/19/2022, Tuesday before midnight
-1 (11/12 pts): 1 day late, 4/25/2022 Monday before midnight
-2 (10/12 pts): 2 days late, 4/26/2022 Tuesday before midnight
(-12/12 pts): non submission, 4/26/2022 Tuesday 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 **<u>the same email attachments</u>** with correct email subject as stated in the email requirement; otherwise, your submission will be rejected.

****
I. Inputs: There are two input files:

a) labelFile (argv[1]): An image file with header

b) propFile argv[2]: the connected component properties.

The format is as below:

- 1st text-line, the header of the input image,
- 2nd text-line is the total number of connected components.
- label-1 // first CC label
- number of pixels
- upperLftR upperLftC //the r c coordinated of the upper left corner
- lowerRgtR lowerRgtC //the r c coordinated of lower right corner
- label-2 // first CC label
- number of pixels
- upperLftR upperLftC //the r c coordinated of the upper left corner
- lowerRgtR lowerRgtC //the r c coordinated of lower right corner

For an example:

```
45  40  0   9  // image header
9                  // there are a total of 9 CCs in the image
1                  // CC label 1
187         // 187 pixels in CC label 1
4     9    // upper left corner of the bounding box at row 4 column 9
35  39  // lower right corner of the bounding box at row 35 column 39
                :                                       :
```

****
II. Outputs: There are two output files

a) chainCode file (Not from argv[])

format: (For easy reading)

numRows numCols minVal maxVal // image header, use one text line

numCC // number of CC in the chainCode

label startRow startCol // use one text line

code1 code2 code3 ....

// All in one text line and with one blank space between codes.

// In real life, each code (0 to 7) only uses 3 bit and without blank spaces between codes!

b) BoundaryFile (not from argv []): a image file with header.

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

- A chainCode class
  - a point struct
    - (int) row
    - (int) col
  - a CCproperty struct
    - (int) label
    - (int) numbpixels
    - (int) minRow, minCol, maxRow, maxCol // bounding box

  - (int) numCC
  - (CCproperty) CC // for storing a connected component properties.
  - (int) numRows
  - (int) numCols
  - (int) minVal
  - (int) maxVal
  - (int **) imageAry // a 2D array to store the label image,
    needs to dynamically allocate at run time (numRows+2 by numCols+2)
  - (int **) boundaryAry // a 2D array to store the reconstructed boundary of objects in the labelled image,
    needs to dynamically allocate at run time (numRows+2 by numCols+2)

- (int **) CCAry // a 2D array to process the chain code of each c.c.
  needs to dynamically allocate at run time (numRows+2 by numCols+2)

- (point) coordOffset [8] // The index is the chain directions from currentP to its eight neighbors;
  // coordOffset[i].row and coordOffset[i].col are the offset of currentP's neighbor at i direction.
  // i.e., coordOffset [ ] are: [(0, +1), (-1, +1), (-1, 0), (-1, -1), (0, -1), (+1, -1), (+1, 0), (+1, +1)]
  // So, given (r, c) of currentP, it neighbors at 0, 1, 2, …, 7 directions would be imgAry (r+0, c+1),
  // imgAry (r-1, c+1),…, imgAry (r+1, c+1). You may *hard code* this offset array.
- (point) neighborCoord [8] // This array store the x-y coordinates of currentP's eight neighbors.
  // This array is very useful for finding the next non-zero neighbor of currentP.
- (int) zeroTable[8] = [6, 0, 0, 2, 2, 4, 4, 6]
  // the index is the direction from currentP to the last zero
  // zeroTable[index] is the direction from nextP to the last zero.
  // You may *hard code* this table as given in the lecture notes.
- (point) startP
- (point) currentP // current none zero border pixel
- (point) nextP // next none-zero border pixel
- (int) lastQ // Range from 0 to 7; it is the direction of the last zero scanned from currentP
- (int) nextDir // the next scanning direction of currentP's neighbors
  // to find nextP, range from 0 to 7, need to mod 8.
- (int) PchainDir // chain code direction from currentP to nextP

- methods:
  - constructor(s)
  - zeroFramed (...) // Reuse code from previous project.
  - loadImage (...) // Read from the label file onto imageAry begin at (1,1)
  - clearCCAry (...) // zero out CCAry
  - loadCCAry (ccLabel) // load the next CC from imageAry of the given label
    // and load the connected component from imageAry to CCAry
    // On your own, you should know how to do this.
  - getChainCode(CC, CCAry)// see algorithm below.
  - loadNeighborsCoord (currentP) // on your own.
    // Given currentP's row and col, the method determines
    // (use coordOffset[] to stores the row and col of each of currentP's 8 neighbors
    //(0 to 7 w.r.t the chain-code direction) in neighborCoord[] array.
  - (int) findNextP(currentP, nextQ, nextP) // see algorithm below.
  - constructBoundary (…) // on your own.
    // Give the chainCode file, create an image contains only the boundary of objects in the labelled file,
  - reformatPrettyPrint (aryTwo, file) // reuse code from your previous project

****
IV. Main (...)
****

Step 0: labelFile ← open label file from argv[]
propFile ← open property file from argv[1]
numRows, numCols, minVal, maxVal ← LabelFile
numRows, numCols, minVal, maxVal ← propFile // need this read, so you may proceed.
numCC ← propFile
imageAry ← dynamically allocated
loadImage (labelFile, imageAry)
CCAry <-- dynamically allocated
Step 1: chainCodeFileName ← argv[1]+ "_chainCode.txt"
BoundaryFileName ← argv[1]+ "_Boundary.txt"

chainCodeFile ← open (chainCodeFileName)
BoundaryFile ← open (BoundaryFileName)

chainCodeFile ← numRows, numCols, minVal, maxVal // image header, one text line
chainCodeFile ← numCC // one text line

Step 2: CC.label ← propFile
CC.numpixels ← propFile
CC.minRow ← propFile
CC.minCol ← propFile
CC.maxRow ← propFile
CC.maxCol ← propFile

Step 3: clearCCAry () // zero out the old CCAry for next CC

Step 4: loadCCAry (CC.label, CCAry) // Extract the pixels with CClabel from imageAry to CCAry.

Step 5: getChainCode (CC, CCAry) // see algorithm below

Step 6: repeat step 2 to step 5 until all connected components are processed.

Step 7: close chainCodeFile

Step 8: reopen chainCodeFile

Step 9: constructBoundary (...)

Step 10: close all files

****
V. getChainCode (CC, CCAry)
****

Step 1: label ← CC.label

Step 2: scan the CCAry from L to R & T to B until
CCAry[iRow, jCol] == label // the beginning of the chain code pixel
ChainCodeFile ← output label, iRow, jCol
startP ← (iRow, jCol)
currentP ← (iRow, jCol)
lastQ ← 4

step 3: nextQ ← mod(lastQ+1, 8)

step 4: PchainDir ← findNextP(currentP, nextQ)
nextP ← neighborCoord [PchainDir]
currentP ←flip the label of currentP in CCAry from positive to negative.

step 5: ChainCodeFile ← output PchainDir follows by a blank

step 6: If PchianDir == 0
lastQ ← zeroTable[7]
else
lastQ ← zeroTable[PchianDir-1]

step 7: currentP ← nextP

step 8: repeat step 3 to step 7 until currentP == startP

****
VI. (int) findNextP (currentP, lastQ)
****

Step 0: loadNeighborsCoord (currentP)
Step 1: index ← lastQ
found ← false
Step 2: iRow ← neighborCoord [index].row
jCol ← neighborCoord [index].col
step 3: if imgAry[iRow][jCol] == label
chainDir ← index
found ← true
Step 4: index ← mod (index+1, 8)
Step 5: repeat step 2 to step 4 until (found == true)
Step 6: return chainDir