Project 8: (C++) You are to implement the Chain code algorithm and reconstruct the object from the resulting chain code. In this project, you may assume that the input binary image contains only one object without holes. This algorithm can apply to image with multiple objects by constructing chain code one object at a time.

There are two (2) components in this project:
1) Image compression: apply chain code algorithm to a binary input file to produce the chain code of the object boundary.
2) Image decompression: reconstruct the object from the chain code
    a) Use the chain code to label the border pixel and mark (use -1) all zero neighbors of border pixels.:
    b) fill the interior pixels between pairs (negative, positive) and (positive, negative)
    c) Turn marked negative pixels to 0.

*** You will be given d3ata files: data1, data2, and data3. Data1 is a simple diamond shape object.
What to do as follows:
1) Implement your program based on the specs given below.
2) Run and debug your program **with data1** until your program produces the reconstructed object that is identical to data1.
3) Run and debug your program **with data2** until your program produces the reconstructed object is identical to data2.
4) Run and debug your program **with data3** until your program produces the reconstructed object is identical to data3.

** Include in your hard copies:
- cover page
- source code
- Pretty Print data1
- Print chainCodeFile for data1
- Print deCompressedFile for data1
- Pretty Print data2
- Print chainCodeFile for data2
- Print deCompressedFile for data2.
- Pretty Print data3
- Print chainCodeFile for data3
- Print deCompressedFile for data3.

****

Language: C++
Project points: 12 pts

Due Date: Soft copy (*.zip) and hard copies (*.pdf):

- +1 (13/12 pts): early submission, 11/25/2022, Friday before midnight
  0 (12/12 pts): on time, 11/30/2022 Wednesday before midnight
  -1 (11/12 pts): 1 day late, 12/1/2022 Thursday before midnight
  -2 (10/12 pts): 2 days late, 12/2/2022 Friday before midnight
  (-12/12 pts): non submission, 12/2/2022 Friday after midnight
  -6/12: does not pass compilation
  0/12: program produces no output

*** Submit your project according to the project submission requirement.

****

I. Input: a binary image (use argv [1]): contain only one object without hole.

****

II. Outputs: There are three outFiles
  a) deBugFile (argv[2]): to print the input image, intermediate results and all debugging prints.
  b) chainCodeFile (not from argv []): To store the chain code of the object in the following format:
    numRows numCols minVal maxVal // image header use one text line
    Label startRow startCol // use one text line
    code1 code2 code3 .... // All in one text line, with one blank space between codes.
    // In real life, each chain code (0 to 7) only use 3 bits and without blank spaces between codes!
  c) deCompressedFile (not from argv []): //An image file to store the reconstructed /decompressed image.

****

III. Data structure:

****

- A chainCode class
    - a point class
        - (int) row
        - (int) col
    - (int) numRows
    - (int) numCols
    - (int) minVal
    - (int) maxVal
    - (int) label // object's label > 0
    - (int **) imgAry // a 2D array to store the input image,
        // needs to dynamically allocate at run time (numRows+2 by numCols+2) and initialized to zero.
    - (int **) reconstructAry // a 2D array needs to dynamically allocate at run time (numRows+2 by numCols+2)
        // to be use for the reconstructed object: the boundary of object and for filled object,

    - (point) coordOffset [8] // coordOffset [ ] are: [(0, +1), (-1, +1), (-1, 0), (-1, -1), (0, -1), (+1, -1), (+1, 0), (+1, +1)]
        // for calculating the row and col positions of a given pixel's 3x3 neighbors. For example, given P(r, c),
        //neighbors at 0 direction would be (r+coordOffset[0].row, c+ coordOffset[0].col)
        //which is imgAry (r+0, c+1); at 1 direction would be (r+coordOffset[1].row, c+ coordOffset[1].col),
        //which is imgAry (r-1, c+1), etc. You may hard code this offSet table.

    - (int) zeroTable [8] = [6, 0, 0, 2, 2, 4, 4, 6] // The look-up table to update the lastZero direction from currentP to
            // nextP. You may *hard code* this table as given in the lecture.
    - (point) startP // the row and col position of the first none zero pixel.
    - (point) currentP // the row and col position of the current none zero border pixel
    - (point) nextP // the row and col position of the next none-zero border pixel
    - (int) lastZero // Range from 0 to 7; it is the direction of the last zero scanned before currentP
    - (int) chainDir // The chain code direction from currentP to nextP

    methods:
    - constructor(s)
    - loadImage (...) // Read from the inFile store onto imgAry, begin at (1,1)
    - getChainCode (…) // see the algorithm below.
    - (int) findNextP (…) // see algorithm below.
    - deCompression (…) // see algorithm below.
    - fillInterior (…) // fill interior object pixels. See algorithm below
    - threshold (…) // turn negative pixels to zero. See algorithm below.
    - Ary2File (ary, file) // output inside frame of ary to file, use reformatPrettyPrint.
    - reformatPrettyPrint (…) // reuse code from your previous project

****

IV. main (...) // Report any bug in any methods below to Dr. Phillips

****

Step 0: inFile, deBugFile ← open input file argv [1] and argv[2]
numRows, numCols, minVal, maxVal ← read from inFile
imgAry, reconstructAry ← dynamically allocated and **initialized to 0** , in effect the boarder are zero framed.

Step 1: chainCodeFileName ← argv [1] + "_chainCode.txt"
deCompressedFileName ← argv [1] + "_deCompressed.txt"

Step 2: chainCodeFile ← open (chainCodeFileName)
deCompressedFile ← open (deCompressedFileName)

Step 3: loadImage (inFile, imgAry)
reformatPrettyPrint (imgAry, deBugFile) // prettyPrint imgAry to deBugFile.

Step 4: getChainCode (imgAry, chainCodeFile) // see algorithm below.

Step 5: close chainCodeFile
Step 6: reopen chainCodeFile

Step 7: decompression (reconstructAry, chainCodeFile, deBugFile)
Step 8: ary2file (reconstructAry, deCompressedFile) // output reconstructAry to deCompressedFile
Step 9: close all files

****

V. getChainCode (imgAry, chainCodeFile)
****

Step 0: chainCodeFile ← Write numRows, numCols, minVal, maxVal to chainCodeFile// one text-line
Step 1: scan imgAry from L to R & T to B
step 2: if imgAry (i, j) > 0 // first none zero pixel
startRow ← i
startCol ← j
chainCodeFile ← output imgAry (i, j), i, j to chainCodeFile // one text-line.
startP.row ←i
startP.col ← j
currentP.row ←i
currentP.col ← j
lastQ ← 4

step 3: lastQ ← mod (lastQ+1, 8)
step 4: chainDir ← findNextP (currentP, lastQ)
Step 5: chainCodeFile ← output chainDir to chainCodeFile // with one space after each code
Step 6: nextP.row ← currentP.row + coordOffset[chainDir].row
nextP.col ← currentP.col + coordOffset[chainDir].col
currentP ← nextP
lastQ ← zeroTable[mod(chainDir-1,8)]

Step 7: repeat step 3 to step 6 until currentP is at (startRow, startCol)

****

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

Step 1: index ← lastQ
found ← false
Step 2: iRow ← currentP.row + coordOffset [index].row
jCol ← currentP.col + coordOffset [index].col

step 3: if imgAry[iRow][jCol] == label
chainDir ← index
found ← true
else index ← mod (index+1, 8)
Step 4: repeat step 2 to step 3 until (found == true)
Step 5: return chainDir

****

VIII. deCompression (reconstructAry, chainCodeFile, debugFile)
****

Step 0: numRows, numCols, minVal, maxVal ← read from chainCodeFile // need to read these but not use them.
Label, startRow startCol ← chainCodeFile

```
        reconstructAry[startRow][startCol] ← Label
        mark ← -1* Label // negative
        point P // declare
        P.row ← startRow
        P.col← startCol
        lastZero ← 4  // chain direction of last visited zero
Step 1:  nextChainCode ← read from chainCodeFile
Step 2:  r ← P.row + coordOffset [lastZero].row
        c ← P.col + coordOffset [lastZero].col
        if (reconstructAry[r][c] ) == 0
                reconstructAry[r][c]← mark
        else deBugFile ← output error message 'something is wrong'

Step 3: lastZero ← mod (lastZero++, 8)
Step 4: repeat step 2 to step 3 while lastZero != nextChainCode
Step 5: // move P to next P
        P.row ← P.row + coordOffset[nextChainCode].row
        P.col ← P.col + coordOffset[nextChainCode].col
        reconstructAry[P.row][ P.col] ← Label
        lastZero ← zeroTable[mod (nextChainCode – 1, 8)]
Step 6: repeat Step 1 to Step 5 until P is at startP
Step 7: ary2file (reconstructAry, deBugFile) // with captions for what you are writing.
Step 8: fillInterior (reconstructAry, Label)
Step 9: ary2file (reconstructAry, deBugFile) // with captions for what you are writing.
Step 10: threshold (reconstructAry)

*******************************
IX. fillInterior (reconstructAry, label)
*******************************
Step 0:  i ← 1
Step 1: for row i
         Assign all zero pixels between adjacent(negative, positive) and adjacent (positive, negative) ←label
        Continue for the next pairs until the end of column of row i
Step 2: i++
Step 3: repeat Step 1 to Step 2 while i < numRows -1

*******************************
X. threshold (reconstructAry)
*******************************
Step 1: process the entire reconstructAry
        if reconstructAry(i, j) < 0
                reconstructAry(i, j) ← 0
```