Project 4 (C++): You are to implement both 4-connected and 8-connected component algorithms as taught in class. Your program will let the user to choose which connectness (4 or 8) to run the program, via argv [2]. In addition, your program gives a conversion option (y or Y for yes, n or N) whether the input data to be converted or not before the processing. (Conversion is to change pixels in an array from 0 to 1 and 1 to 0.)

*** You will be given two data files, data1 and data2, and the answer file of data1.
What you need to do as follows:
a) Implement your program based on the specs given below.
b) Test and debug your program using data1 for 8-connected **with option N** until it produces the same result as given in the answer file.
c) Test and debug your program using data1 for 4-connected **with option N** until it produces the same result as given in the answer file.
d) Run your program using data2 for 8-connected **with option N**. (Eyeball the result for correctness.)
e) Run your program using data2 for 4-connected **with option Y**. (Eyeball the result for correctness and See if you know the meaning of the result in e).
** On each run, your program will produce three files: RFprettyPrintFile, LabelFile, and propertyFile.
** labelFile and propertyFile will be used as input in your future project(s).

Your hard copies include:

- Cover page
- Source code
- RFprettyPrintFile for 8-connectness run on data1
- labelFile for 8-connectness run on data1
- propertyFile for 8-connectness run on data1
- deBugFile // limited to 2 pages if more than 2.

- RFprettyPrintFile for 4-connectness run on data1
- labelFile for 4-connectness run on data1
- propertyFile for 4-connectness run on data1
- deBugFile // limited to 2 pages if more than 2.

- RFprettyPrintFile for 8-connectness run on data2
- labelFile for 8-connectness run on data2
- propertyFile for 8-connectness run on r data2
- deBugFile // limited to 2 pages if more than 2.

- RFprettyPrintFile for 4-connectness run on data2 after conversion.
- labelFile for 4-connectness run on data2 after conversion.
- propertyFile for 4-connectness run on data2 after conversion.
- deBugFile // limited to 2 pages if more than 2.

****
Language: C++
Project points:12 pts
Due Date: Soft copy (*.zip) and hard copies (*.pdf):

+1 (13/12 pts): +1 for early submission, 3/15/2023, Wednesday before midnight
-0 (12/12 pts): on time, 3/19/2023 Sunday before midnight
(-12/12 pts): non-submission, 3/19/2023 Sunday after midnight (NO LATE SUBMISSION!)

*** 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 **the same email attachments** with correct email subject as stated in the email requirement; otherwise, your submission will be rejected.
****

I. Inputs:

a) inFile (argv [1]): A binary image.
b) Connectness (argv [2]): 4 for 4-connectness, 8 for 8-connectness.
c) conversion (argv [3]): y or Y for yes, n or N for no.

II. Outputs:

a) RFprettyPrintFile (argv [4]): (include in your hard copy) for the followings:
** a proper caption means the caption should say what the printing is.

- reformatPrettyPrint of the result of the Pass-1 with proper captions
- print newLabel and the EQAry after Pass-1, with proper captions
- reformatPrettyPrint of the result of the Pass-2 with proper captions
- print newLabel and the EQAry after Pass-2, with proper captions
- Print the EQAry after manage the EQAry, with proper caption
- reformatPrettyPrint of the result of the Pass-3 with proper captions
- reformatPrettyPrint of the result bounding boxes drawing.

b) labelFile (argv [5]): to store the connected component labels of Pass-3 -- the labelled image file with image header, numRows numCols newMin NewMax.
** This file to be used in future processing.

c) propertyFile (argv [6]): To store the connected component properties.
*** This file to be used in future processing.
The format is to be as below:

- 1st text-line, the header of the input image,
- 2nd text-line is the total number of connected components.
- 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
- number of pixels
- minR, minC //the r c coordinated of the upper left corner
- maxR, maxC //the r c coordinated of lower right corner

For an example:

```
45  40  0   9  // image header
9               // indicates 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
2                   // CC label 2
36          // 36 pixels in CC label 2
14     19    // upper left corner of the bounding box at row 14 column 19
25  49  // lower right corner of the bounding box at row 25 column 49
```

:

d) deBugFile (argv [7]) : for all debugging prints in the program.

****

III. Data structure:

****

- A Property (1D struct)
  - (int) label // The component label
  - (int) numPixels // total number of pixels in the cc.
  - (int) minR // with respect to the input image.
  - (int) minC // with respect to the input image.
  - (int) maxR // with respect to the input image.
  - (int) maxC // with respect to the input image.

  // In the Cartesian coordinate system, any rectangular box can be represented by two points: upper-left corner and the lower-right of the box. Here, the two points:(minR minC) and (maxR maxC) represents the smallest rectangular box that a cc can fit in the box; object pixels can be on the border of the box.
- A ccLabel class
  - (int) numRows
  - (int) numCols
  - (int) minVal
  - (int) maxVal
  - (int) newLabel // initialize to 0
  - (int) trueNumCC // the true number of connected components in the image

- (int) newMin // set to 0
- (int) newMax // set to trueNumCC
- (int **) zeroFramedAry // a 2D array of size numRows + 2 by numCols + 2, dynamically allocate at run time
- (int) NonZeroNeighborAry [5] // 5 is the max number of neighbors you have to check. For easy programming,
//you may consider using this 1-D array to store pixel (i, j)'s non-zero neighbors during pass 1 and pass2.
- (int *) EQAry // a 1-D array, of size (numRows * numCols) / 4
// dynamically allocate at run time, and initialize to its index, i.e., EQAry[i] = i.
- (char) option // the option for conversion.
- (Property *) CCproperty // A struct 1D array (the size is the trueNumCC+1) to store components' properties.
// dynamically allocate at runtime.

- methods:
- constructor(...) // need to dynamically allocate all arrays; and assign values to numRows, etc.
- zero2D (...) // ** Initialized a 2-D array to zero. You must implement this method.
- negative1D (...) // ** Initialized a 1-D array to -1.
- loadImage (...) // read from input file and write to zeroFramedAry begin at (1,1)
- conversion (…) // converts every pixel inside the zeroFramedAry begin at (1,1) from 0 to 1 and 1 to zero.
// leave the frame boarder to 0.
- imgReformat (zeroFramedAry, RFprettyPrintFile)
// Print zeroFramedAry to RFprettyPrintFile. Reuse code from your previous project.
- connect8Pass1 (...) // On your own, algorithm was presented in class.
- connect8Pass2 (...) // On your own, algorithm was presented in class.
- connect4Pass1 (...) // On your own, algorithm was presented in class.
- connect4Pass2 (...) // On your own, algorithm was presented in class.
- connectPass3 (...) // See algorithm below.
- updateEQ (...) // Update EQAry for all non-zero neighbors to minLabel.
// In case 3 of the pass1 and pass2 of 4-conn and 8-conn method, the method needs to update EQAry for
// those non-minimum label of neighbors of p(i, j) to minLabel; It will be easier to use
//NonZeroNeighborAry, at first to store all non-zero neighbors of p(i, j) in ascending order to find
// minLabel and update EQ table.
- (int) manageEQAry (...) // The algorithm was given in class.
// The method returns the true number of CCs in the labelled image.
- printCCproperty (...) // Prints the component properties to propertyFile using the format given in the above.
- printEQAry (...) // Print EQAry with index up to newLabel, not beyond. On your own
- drawBoxes (...) // Draw the bounding boxes of CC in zeroFramedAry. See algorithm below
- printImg (...) // Output image header and zeroFramedAry (inside of framing) to labelFile. On your own.

****
IV. main(...)
****
step 0: inFile ← open the input file from argv [1]
Connectness ← argv [2]
option ← argv [3]
RFprettyPrintFile, labelFile, propertyFile, deBugFile ← open from argv []
numRows, numCols, minVal, maxVal ← read from inFile
zeroFramedAry ← dynamically allocate.
newLabel ← 0
step 1: zero2D (zeroFramedAry)
step 2: loadImage (inFile, zeroFramedAry)
step 3: if option == 'y' or 'Y'
conversion (zeroFramedAry)
step 4: if connectness == 4
connected4 (zeroFramedAry, newLabel, EQAry, RFprettyPrintFile, deBugFile)
step 5: if connectness == 8
connected4 (zeroFramedAry, newLabel, EQAry, RFprettyPrintFile, deBugFile)
step 6: labelFile ← output numRows, numCols, newMin, newMax to labelFile
step 7: printImg (zeroFramedAry, labelFile) // Output the result of pass3 inside of zeroFramedAry

step 8: printCCproperty (propertyFile) // print cc properties to propertyFile
step 9: drawBoxes (zeroFramedAry, CCproperty, trueNumCC) // draw on zeroFramed image.
step 10: imgReformat (zeroFramedAry, RFprettyPrintFile)
step 11: print trueNumCC to RFprettyPrintFile with proper caption
step 12: close all files

****
V. connected4 (zeroFramedAry, newLabel, EQAry, RFprettyPrintFile, deBugFile)
****
Step 0: deBugFile ← "entering connected4 method"
Step 1: connect4Pass1 (zeroFramedAry, newLabel, EQAry)
    deBugFile ← "After connected4 pass1, newLabel =" // print newLable
    imgReformat (zeroFramedAry, RFprettyPrintFile)
    printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLable with proper caption

Step 2: Connect4Pass2 (zeroFramedAry, EQAry)
    deBugFile ← "After connected4 pass2, newLabel =" // print newLable
    imgReformat (zeroFramedAry, RFprettyPrintFile)
    printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLabel with proper caption

Step 3: trueNumCC ← manageEQAry (EQAry, newLabel)
    printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLabel with proper caption
    newMin ← 0
    newMax ← trueNumCC
    CCproperty ← dynamically allocate size of trueNumCC+1
    deBugFile ← "In connected4, after manage EQAry, trueNumCC =" // print trueNumCC

Step 4: connectPass3 (zeroFramedAry, EQAry, CCproperty, trueNumCC, deBugFile) // see algorithm below.
Step 5: imgReformat (zeroFramedAry, RFprettyPrintFile)
Step 6: printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLabel with proper caption
Step 7: deBugFile ← "Leaving connected4 method"

****
VI. connected8 (zeroFramedAry, newLabel, EQAry, RFprettyPrintFile, deBugFile)
****
Step 0: deBugFile ← "entering connected8 method"
Step 1: connect8Pass1 (zeroFramedAry, newLabel, EQAry)
    deBugFile ← "After connected8 pass1, newLabel =" // print newLable
    imgReformat (zeroFramedAry, RFprettyPrintFile)
    printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLable with proper caption

Step 2: Connect8Pass2 (zeroFramedAry, EQAry)
    deBugFile ← "After connected8 pass2, newLabel =" // print newLable
    imgReformat (zeroFramedAry, RFprettyPrintFile)
    printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLabel with proper caption

Step 3: trueNumCC ← manageEQAry (EQAry, newLabel)
    printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLabel with proper caption
    newMin ← 0
    newMax ← trueNumCC
    CCproperty ← dynamically allocate size of trueNumCC+1
    deBugFile ← "In connected8, after manage EQAry, trueNumCC =" // print trueNumCC

Step 4: connectPass3 (zeroFramedAry, EQAry, CCproperty, trueNumCC, deBugFile) // see algorithm below.
Step 5: imgReformat (zeroFramedAry, RFprettyPrintFile)
Step 6: printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLabel with proper caption
Step 7: deBugFile ← "Leaving connected8 method"

****
VII. connectPass3 (zeroFramedAry, EQAry, CCproperty, trueNumCC, deBugFile)
****
Step 0: deBugFile ← "entering connectPas3 method"
Step 1: for i = 1 to trueNumCC
CCproperty[i].label ← i
CCproperty[i].numPixels ← 0
CCproperty[i].minR ← numRow
CCproperty[i].maxR ← 0
CCproperty[i].minC ← numCol
CCproperty[i].maxC ← 0

Step 2: scan inside of the zeroFramedAry left-right & top-bottom
p(r, c) ← next pixel

Step 3: if p(r, c) > 0
zeroFramedAry [r, c] ← EQAry[p(r, c)] // relabeling.
k ← zeroFramedAry [r, c]
CCproperty[k].numPixels++
if r < CCproperty[k].minR
CCproperty[k].minR ← r
if r > CCproperty[k].maxR
CCproperty[k].maxR ← r
if c < CCproperty[k].minC
CCproperty[k].minC ← c
if c > CCproperty[k].maxC
CCproperty[k].maxC ← c

Step 4: repeat Step 2 to Step 3 until all pixels inside of zeroFramedAry are processed
Step 5: deBugFile ← "leaving connectPas3 method"

****
VIII. drawBoxes (zeroFramedAry, CCproperty, trueNumCC)
****
step 1: index ← 1
step 2: minRow ← CCproperty[index]'s minR + 1
minCol ← CCproperty[index]'s minC + 1
maxRow ← CCproperty[index]'s maxR + 1
maxCol ← CCproperty[index]'s maxC + 1
label ← CCproperty[index]'s label

step 3: Assign label to all pixels on minRow of zeroFramedAry, from minCol to maxCol ← label //use a loop.
Assign label to all pixels on maxRow of zeroFramedAry, from minCol to maxCol ← label //use a loop.
Assign label to all pixels on minCol of zeroFramedAry, from minRow to maxRow ← label //use a loop.
Assign label to all pixels on maxCol of zeroFramedAry, from minRow to maxRow ← label //use a loop.

step 4: index++
step 5: repeat step 2 to step 4 while index <= trueNumCC