Student: Michael Grossman

Due Date: 5/19/2022

**Code**

#include <fstream>

#include <iostream>

using namespace std;

class EdgeDetector{

    public:

    int numRows, numCols, minVal, maxVal;

    int\*\* MFAry;

    int\*\* tempAry;

    int\*\* edgeAry;

    int\*\* maskAry;

    void run(ifstream& inp, ifstream& mask, ofstream& outp1, ofstream& outp2);

    int convolve(int\*\* inp, int r, int c, int\*\* m);

    void binaryEdge(int\*\* fromAry, int\*\* toAry);

};

int main(int argv, char\*\* args){

    string input\_filename = args[1];

    string mask\_filename = args[2];

    string output\_filename1 = args[3];

    string output\_filename2 = args[4];

    ifstream inp(input\_filename);

    ifstream mask(mask\_filename);

    ofstream outp1(output\_filename1);

    ofstream outp2(output\_filename2);

    EdgeDetector ed;

    ed.run(inp, mask, outp1, outp2);

    inp.close();

    mask.close();

    outp1.close();

    outp2.close();

}

void EdgeDetector::run(ifstream& inp, ifstream& mask, ofstream& outp1, ofstream& outp2){

    inp >> numRows; inp >> numCols; inp >> minVal; inp >> maxVal;

    //dynamically allocate

    MFAry = new int\*[numRows+2];

    tempAry = new int\*[numRows+2];

    edgeAry = new int\*[numRows+2];

    for(int i = 0; i < numRows + 2; ++i){

        MFAry[i] = new int[numCols+2]{0};

        tempAry[i] = new int[numCols+2]{0};

        edgeAry[i] = new int[numCols+2]{0};

    }

    maskAry = new int\*[3];

    for(int i = 0; i < 3; ++i){

        maskAry[i] = new int[3]{0};

    }

    //load image into MFAry

    for(int row = 1; row <= numRows; ++row){

        for(int col = 1; col <= numCols; ++col){

            inp >> MFAry[row][col];

        }

    }

    //mirror frame top and bottom

    for(int col = 1; col <= numCols; ++col){

        MFAry[0][col] = MFAry[1][col];

        MFAry[numRows+1][col] = MFAry[numRows][col];

    }

    //mirror frame left and right

    for(int row = 1; row <= numRows; ++row){

        MFAry[row][0] = MFAry[row][1];

        MFAry[row][numCols+1] = MFAry[row][numCols];

    }

    //mirror corners -- over axis

    MFAry[0][0] = MFAry[1][1];

    MFAry[0][numCols+1] = MFAry[1][numCols];

    MFAry[numRows+1][0] = MFAry[numRows][1];

    MFAry[numRows+1][numCols+1] = MFAry[numRows][numCols];

    //load 3x3 Mask

    for(int i = 0; i < 3; ++i){

        for(int j = 0; j < 3; ++j){

            mask >> maskAry[i][j];

            //cout << maskAry[i][j] << " ";

        }

    }

    //loading convolutions into tempAry

    for(int row = 1; row <= numRows; ++row){

        for(int col = 1; col <= numCols; ++col){

            tempAry[row][col] = convolve(MFAry, row, col, maskAry);

        }

    }

    //thresholding tempAry

    for(int row = 1; row <= numRows; ++row){

        for(int col = 1; col <= numCols; ++col){

            if(tempAry[row][col] > 0) tempAry[row][col] = 1;

            else tempAry[row][col] = 0;

        }

    }

    //output thresholded tempAry

    for(int row = 1; row <= numRows; ++row){

        for(int col = 1; col <= numCols; ++col){

            if (tempAry[row][col] > 0){

             outp1 << tempAry[row][col];

            }

            else outp1 << ".";

            outp1 << " ";

        }

        outp1 << "\n";

    }

    //zero-crossing edge detection

    binaryEdge(tempAry, edgeAry);

    //output edgeAry

    for(int row = 1; row <= numRows; ++row){

        for(int col = 1; col <= numCols; ++col){

            if(edgeAry[row][col] > 0){

                outp2 << edgeAry[row][col];

            }

            else outp2 << ".";

            outp2 << " ";

        }

        outp2 << "\n";

    }

}

int EdgeDetector::convolve(int\*\* inp, int r, int c, int\*\* m){

    int retVal = 0;

    for(int row = r-1; row <= r+1; ++row){

        for(int col = c-1; col <= c+1; col++){

            retVal += inp[row][col] \* m[row-r+1][col-c+1];

        }

    }

    return retVal;

}

void EdgeDetector::binaryEdge(int\*\* fromAry, int\*\* toAry){

    for(int row = 1; row <= numRows; ++row){

        for(int col = 1; col <= numCols; ++col){

            if(fromAry[row][col] == 1 && (fromAry[row][col-1] == 0 || fromAry[row][col+1] == 0)){

                toAry[row][col] = 1;

            }

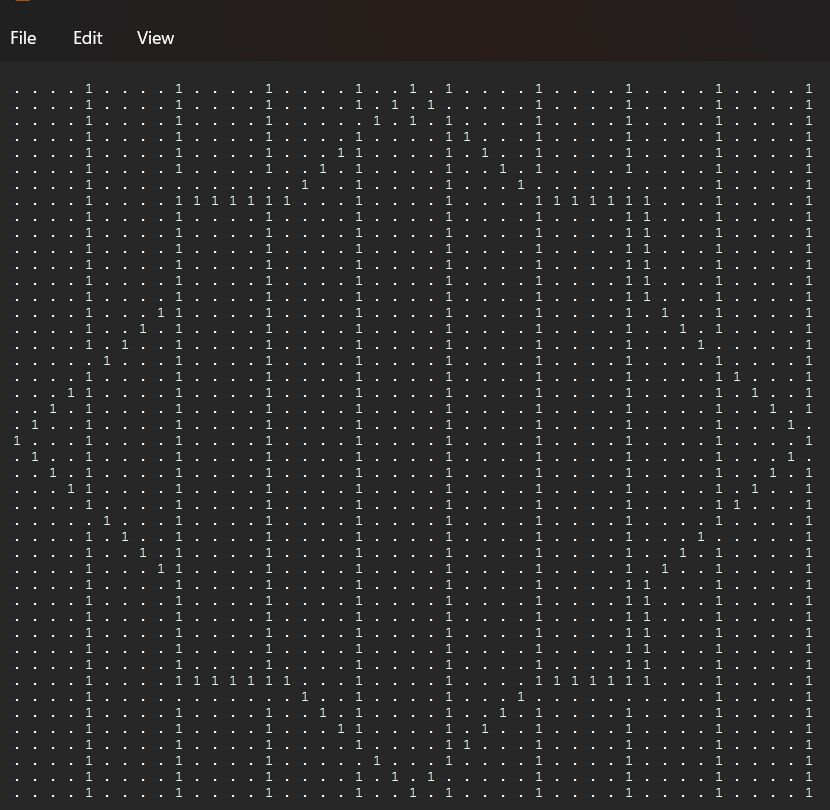
            else toAry[row][col] = 0;

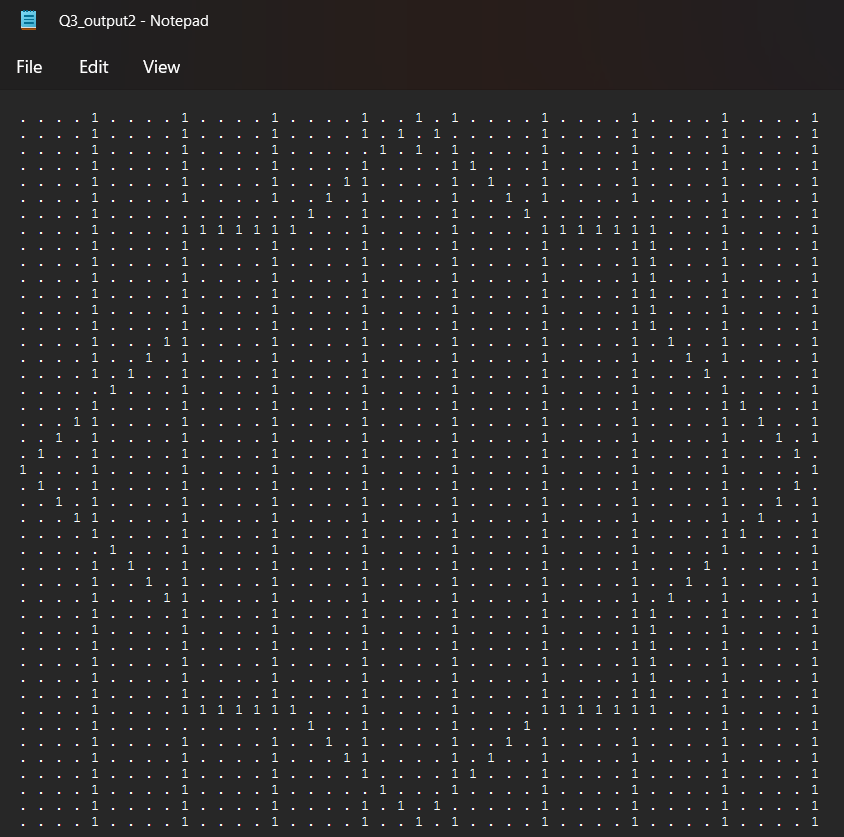
        }

    }

}

**OUTPUTS (next page)**

****

****