CV Project 7: Chain Code CPP

Name: Michael Grossman

Due Date: 4/24/2022

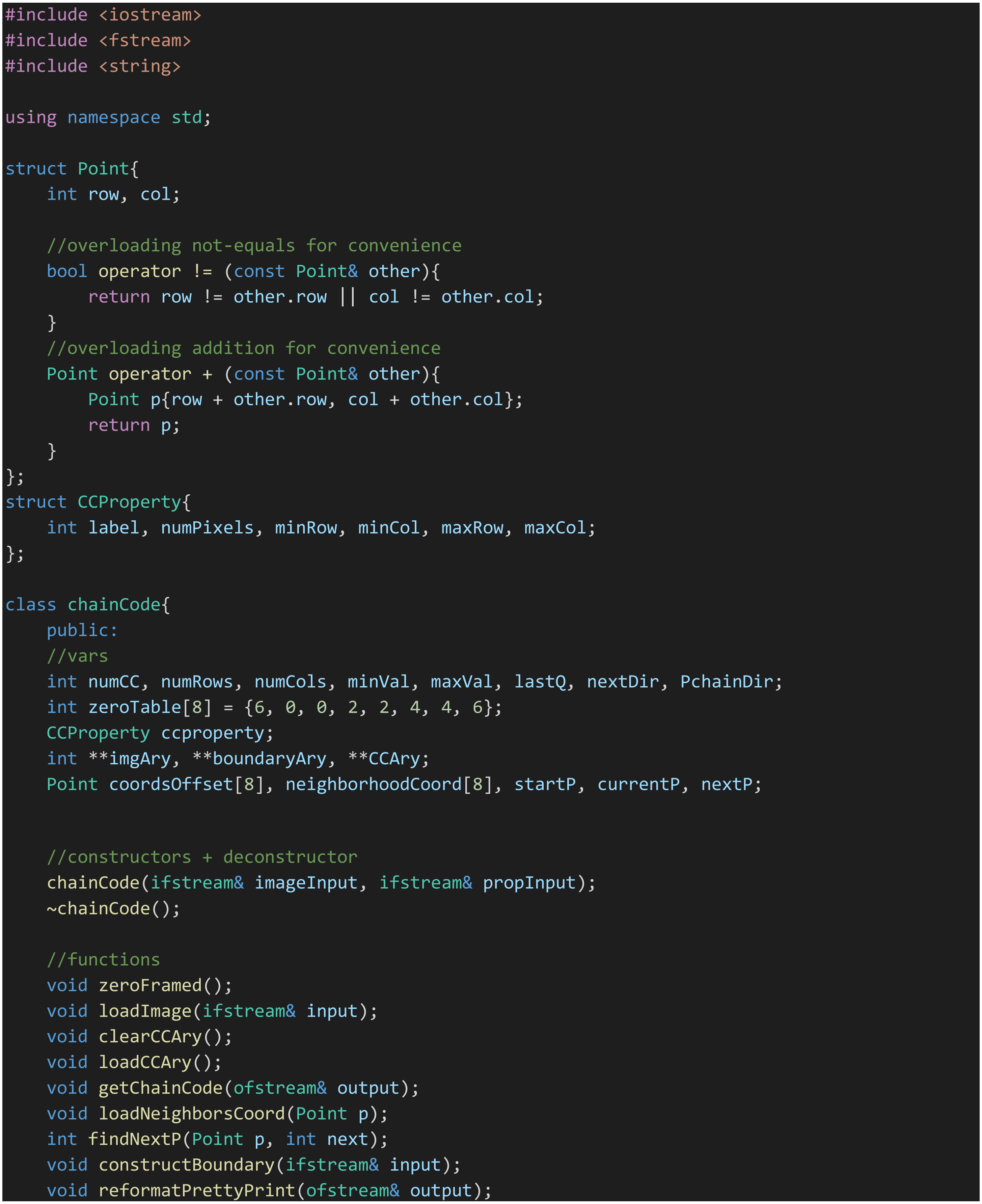
Algorithmic steps for obtaining the chain code of an object given connected component data on the object, and image containing the object.

```
1.  Label ← connected_component.Label
2.  Scan the image L to R, & T to B until $P_{ij}$, the pixel at row i and column j, equals Label
3.  Chain_Code_output ← Label, i, j
4.  startP ← (i, j)
5.  currentP ← (i, j)
6.  lastQ ← 4
7.  nextQ ← mod(lastQ + 1, 8)
8.  PchainDir ←findNextP(currentP, nextQ)
9.  nextP ←neighborhoodCoord[PchainDir]
10. Chain_Code_output ← PchainDir and a space
11. If PchainDir == 0:
12. lastQ ←zeroTable[7]
13. Else:
14. lastQ ←zeroTable[PchainDir - 1]
15. end if-else
16. currentP ← nextP
17. repeat steps 7 to 16 until currentP == startP
```

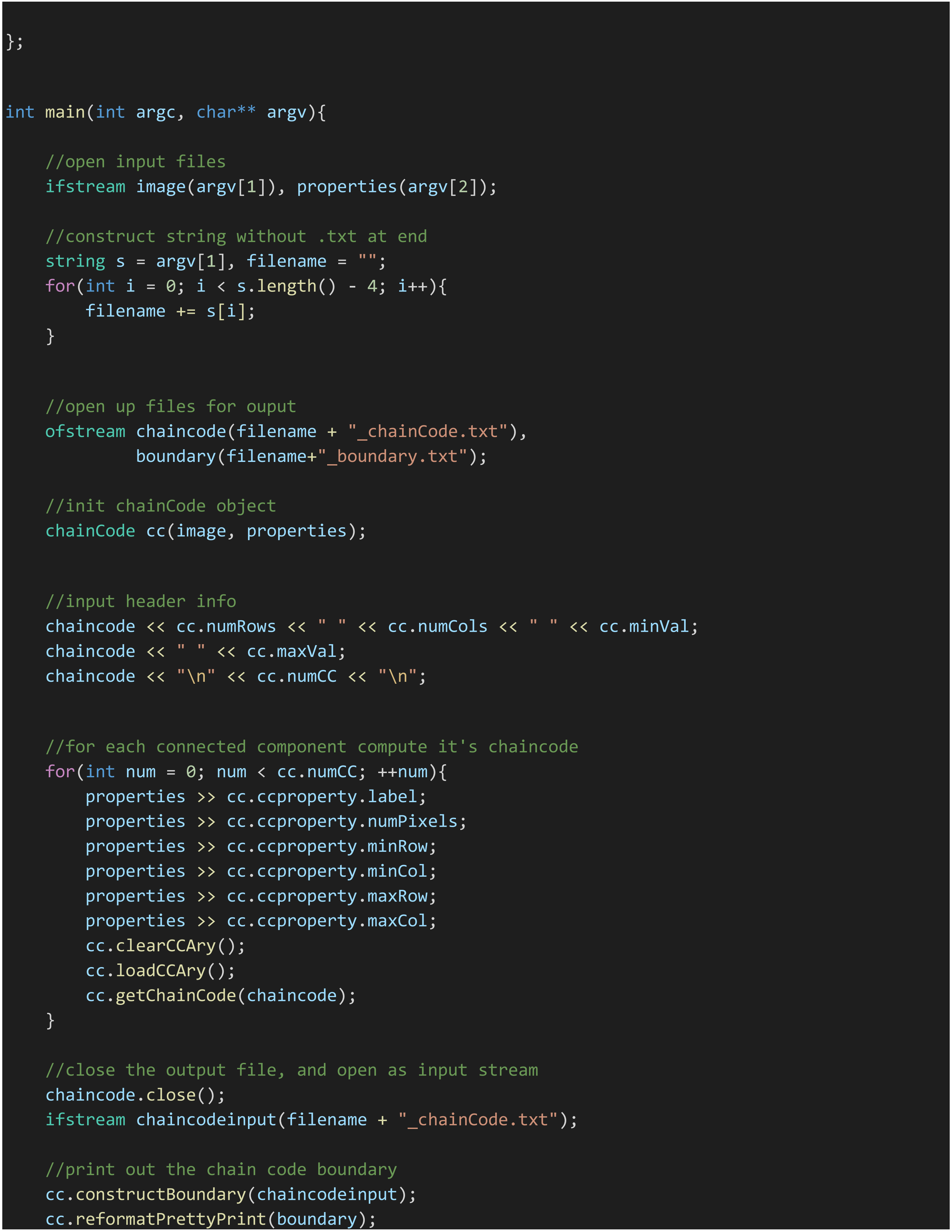
Algorithmic steps for finding the next point for the chain code algorithm given a current point, last point, and an image containing the object being operated on:

```
1.  loadNeighborhoodCoords(currentP)
2.  index ←lastQ
3.  found = false
4.  i ← neighbordhoodCoord[index].row
5.  j ← neighbordhoodCoord[index].col
6.  if pixel $P_{ij}$ == label:
7.  chainDir ← index
8.  found = true
9.  end-if
10. index ← mod(index + 1, 8)
11. repeat4 to 10 until found == true
12. return chainDir
```

CODE

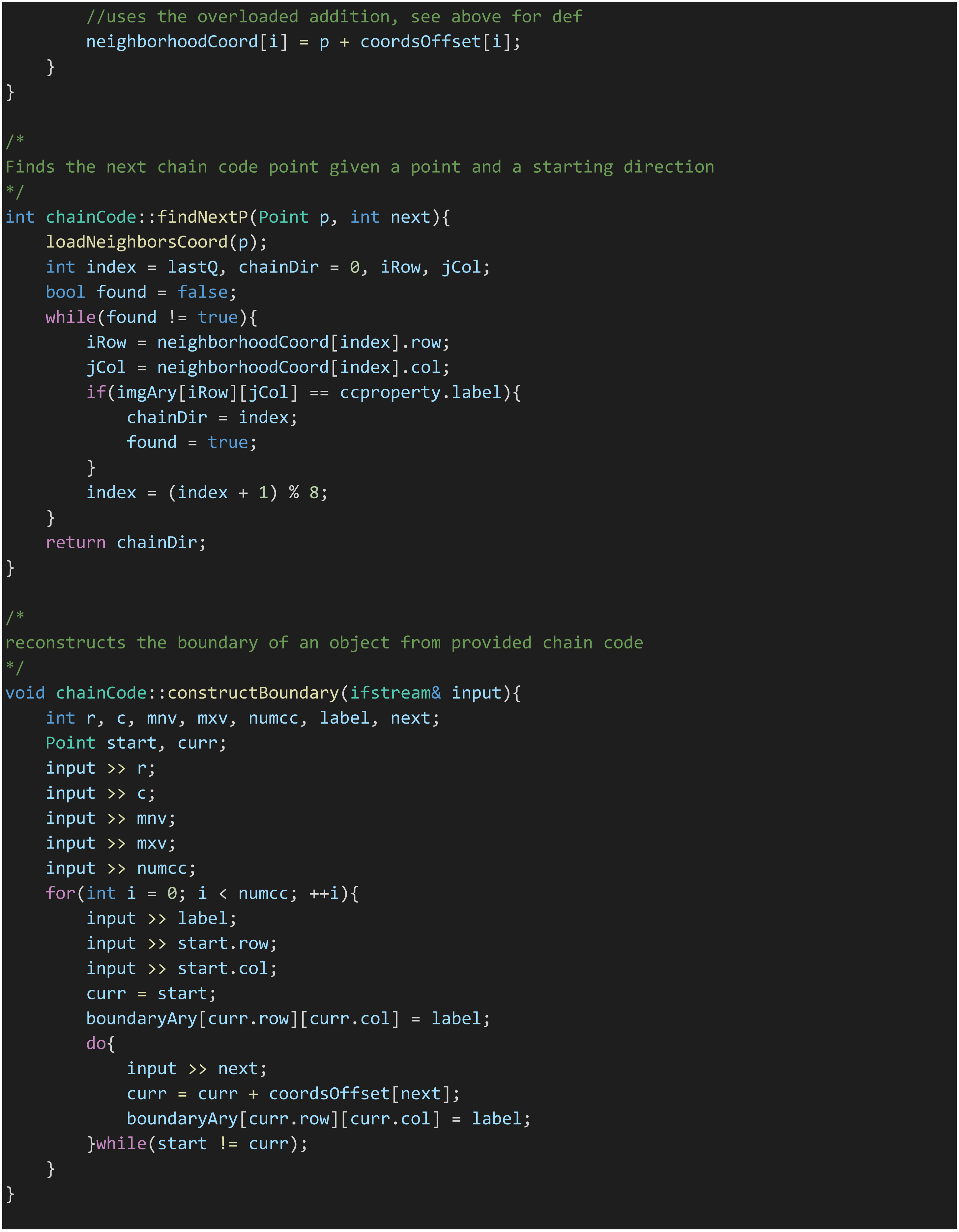


```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

struct Point{
    int row, col;

    //overloading not-equals for convenience
    bool operator != (const Point& other){
        return row != other.row || col != other.col;
    }
    //overloading addition for convenience
    Point operator + (const Point& other){
        Point p{row + other.row, col + other.col};
        return p;
    }
};
struct CCProperty{
    int label, numPixels, minRow, minCol, maxRow, maxCol;
};

class chainCode{
    public:
    //vars
    int numCC, numRows, numCols, minVal, maxVal, lastQ, nextDir, PchainDir;
    int zeroTable[8] = {6, 0, 0, 2, 2, 4, 4, 6};
    CCProperty ccproperty;
    int **imgAry, **boundaryAry, **CCAry;
    Point coordsOffset[8], neighborhoodCoord[8], startP, currentP, nextP;


    //constructors + deconstructor
    chainCode(ifstream& imageInput, ifstream& propInput);
    ~chainCode();

    //functions
    void zeroFramed();
    void loadImage(ifstream& input);
    void clearCCAry();
    void loadCCAry();
    void getChainCode(ofstream& output);
    void loadNeighborsCoord(Point p);
    int findNextP(Point p, int next);
    void constructBoundary(ifstream& input);
    void reformatPrettyPrint(ofstream& output);
```

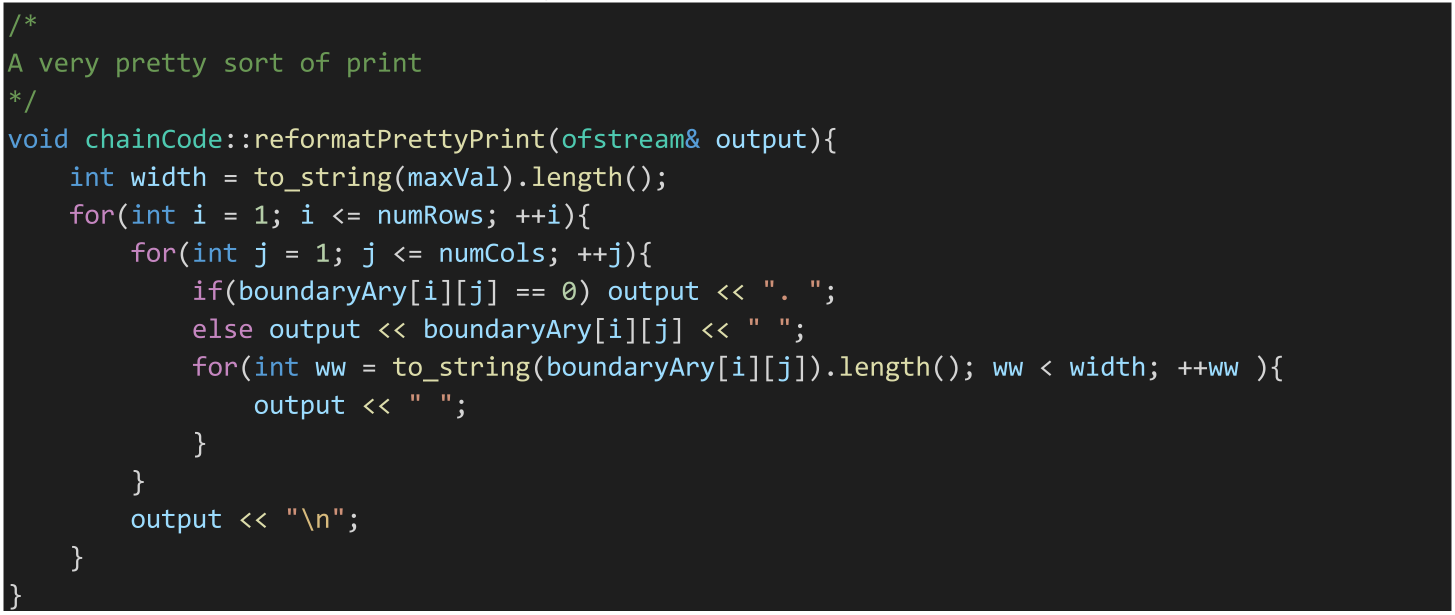


```cpp
};


int main(int argc, char** argv){

    //open input files
    ifstream image(argv[1]), properties(argv[2]);

    //construct string without .txt at end
    string s = argv[1], filename = "";
    for(int i = 0; i < s.length() - 4; i++){
        filename += s[i];
    }


    //open up files for ouput
    ofstream chaincode(filename + "_chainCode.txt"),
             boundary(filename+"_boundary.txt");

    //init chainCode object
    chainCode cc(image, properties);


    //input header info
    chaincode << cc.numRows << " " << cc.numCols << " " << cc.minVal;
    chaincode << " " << cc.maxVal;
    chaincode << "\n" << cc.numCC << "\n";


    //for each connected component compute it's chaincode
    for(int num = 0; num < cc.numCC; ++num){
        properties >> cc.ccproperty.label;
        properties >> cc.ccproperty.numPixels;
        properties >> cc.ccproperty.minRow;
        properties >> cc.ccproperty.minCol;
        properties >> cc.ccproperty.maxRow;
        properties >> cc.ccproperty.maxCol;
        cc.clearCCAry();
        cc.loadCCAry();
        cc.getChainCode(chaincode);
    }

    //close the output file, and open as input stream
    chaincode.close();
    ifstream chaincodeinput(filename + "_chainCode.txt");

    //print out the chain code boundary
    cc.constructBoundary(chaincodeinput);
    cc.reformatPrettyPrint(boundary);
    //close all files
    chaincodeinput.close();
    boundary.close();
    image.close();
    properties.close();

}

/*
Constructor. reads in the variables from the given files, dynamically allocated
space to store the image, and work on the image, and then loads the image in
*/
chainCode::chainCode(ifstream& imageInput, ifstream& propInput){
    imageInput >> numRows;
    propInput >> numRows;
    imageInput >> numCols;
    propInput >> numCols;
    imageInput >> minVal;
    propInput >> minVal;
    imageInput >> maxVal;
    propInput >> maxVal;
    propInput >> numCC;

    imgAry = new int*[numRows+2];
    CCAry = new int*[numRows+2];
    boundaryAry = new int*[numRows+2];
    for(int i = 0; i < numRows + 2; ++i){
        imgAry[i] = new int[numCols + 2];
        CCAry[i] = new int[numCols + 2];
        boundaryAry[i] = new int[numCols + 2];
    }

    coordsOffset[0] = {0, 1};
    coordsOffset[1] = {-1, 1};
    coordsOffset[2] = {-1, 0};
    coordsOffset[3] = {-1, -1};
    coordsOffset[4] = {0, -1};
    coordsOffset[5] = {1, -1};
    coordsOffset[6] = {1, 0};
    coordsOffset[7] = {1, 1};

    loadImage(imageInput);
}

/*
Deconstructor. Deallocated the dynamically allocated arrays
*/
chainCode::~chainCode(){
    for(int i = 0; i < numRows + 2; ++i){
        delete[] imgAry[i];
        delete[] CCAry[i];
        delete[] boundaryAry[i];
    }
    delete[] imgAry;
    delete[] CCAry;
    delete[] boundaryAry;
}

/*
Add zero's to all locations - thus framing with zero, always
called before the image is loaded
*/
void chainCode::zeroFramed(){
    for(int i = 0; i < numRows + 2; ++i){
        for(int j = 0; j < numCols + 2; ++j){
            imgAry[i][j] = 0;
            CCAry[i][j] = 0;
            boundaryAry[i][j] = 0;
        }
    }
}

/*
Add a zero frame to the iamge and then load the image
to the inside of the frame
*/
void chainCode::loadImage(ifstream& input){
    zeroFramed();
    for(int i = 1; i <= numRows; ++i){
        for(int j = 1; j <= numCols; ++j){
            input >> imgAry[i][j];
        }
    }
}

/*
Zero out the entire CCAry
*/
void chainCode::clearCCAry(){
    for(int i = 1; i <= numRows; ++i){
        for(int j = 1; j <= numCols; ++j){
            CCAry[i][j] = 0;
        }
    }
}

/*
```

```
Loads in a single component to work on from the stored image
*/
void chainCode::loadCCAry(){
    for(int i = ccproperty.minRow; i <= ccproperty.maxRow; ++i){
        for(int j = ccproperty.minCol; j <= ccproperty.maxCol; ++j){
            if(imgAry[i][j] == ccproperty.label){
                CCAry[i][j] = ccproperty.label;
            }
        }
    }
}

/*
Searches for the first pixel, and then creates a chain code from
that point, moves counter-clockwise around the boarder of the object.
*/
void chainCode::getChainCode(ofstream& output){
    bool found = false;
    for(int i = ccproperty.minRow; i <= ccproperty.maxRow && !found; ++i){
        for(int j = ccproperty.minCol; j <= ccproperty.maxCol && !found; ++j){
                if(CCAry[i][j] == ccproperty.label){
                    startP.row = i;
                    startP.col = j;
                    currentP.row = i;
                    currentP.col = j;
                    lastQ = 4;
                    found = true;
                }
        }
    }
    output << ccproperty.label << " " << startP.row << " " <<startP.col << "\n";
    do{
        nextDir = (lastQ + 1) % 8;
        PchainDir = findNextP(currentP, nextDir);
        nextP = neighborhoodCoord[PchainDir];
        //did not need to negate the pixel, serves no purpose
        output << PchainDir << " ";
        if(PchainDir == 0) lastQ = zeroTable[7];
        else lastQ = zeroTable[PchainDir - 1];
        currentP = nextP;
    }while(currentP != startP); //while you still have not made a full lap
    output << "\n";
}

/*
loads in the neighborhood of the given pixel
*/
void chainCode::loadNeighborsCoord(Point p){
    for(int i = 0; i < 8; ++i){
```



```
        //uses the overloaded addition, see above for def
        neighborhoodCoord[i] = p + coordsOffset[i];
    }
}

/*
Finds the next chain code point given a point and a starting direction
*/
int chainCode::findNextP(Point p, int next){
    loadNeighborsCoord(p);
    int index = lastQ, chainDir = 0, iRow, jCol;
    bool found = false;
    while(found != true){
        iRow = neighborhoodCoord[index].row;
        jCol = neighborhoodCoord[index].col;
        if(imgAry[iRow][jCol] == ccproperty.label){
            chainDir = index;
            found = true;
        }
        index = (index + 1) % 8;
    }
    return chainDir;
}

/*
reconstructs the boundary of an object from provided chain code
*/
void chainCode::constructBoundary(ifstream& input){
    int r, c, mnv, mxv, numcc, label, next;
    Point start, curr;
    input >> r;
    input >> c;
    input >> mnv;
    input >> mxv;
    input >> numcc;
    for(int i = 0; i < numcc; ++i){
        input >> label;
        input >> start.row;
        input >> start.col;
        curr = start;
        boundaryAry[curr.row][curr.col] = label;
        do{
            input >> next;
            curr = curr + coordsOffset[next];
            boundaryAry[curr.row][curr.col] = label;
        }while(start != curr);
    }
}
```



```cpp
/*
A very pretty sort of print
*/
void chainCode::reformatPrettyPrint(ofstream& output){
    int width = to_string(maxVal).length();
    for(int i = 1; i <= numRows; ++i){
        for(int j = 1; j <= numCols; ++j){
            if(boundaryAry[i][j] == 0) output << ". ";
            else output << boundaryAry[i][j] << " ";
            for(int ww = to_string(boundaryAry[i][j]).length(); ww < width; ++ww ){
                output << " ";
            }
        }
        output << "\n";
    }
}
```

OUTPUT

```
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . 1 1 1 . . . . . . . . . . . . . .
. . . . . . . . . . . . . 1 . . . 1 . . . . . . . . . . . . .
. . . . . . . . . . . . 1 . . . . . 1 . . . . . . . . . . . .
. . . . . . . . . . . 1 . . . . . . . 1 . . . . . . . . . . .
. . . . . . . . . . 1 . . . . . . . . . 1 . . . . . . . . . .
. . . . . . . . . 1 . . . . . . . . . . . 1 . . . . . . . . .
. . . . . . . . . 1 1 1 1 1 1 . 1 1 1 1 1 1 . . . . . . . . .
. . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . .
. . . . . . . . . 1 1 1 1 1 1 . 1 1 1 1 1 1 . . . . . . . . .
. . . . . . . . . 1 . . . . . . . . . . . 1 . . . . . . . . .
. . . . . . . . . . 1 1 . . . . . . . 1 1 . . . . . . . . . .
. . . . . . . . . . . . 1 . . . . . 1 . . . . . . . . . . . .
. . . . . . . . . . . . . 1 . . . 1 . . . . . . . . . . . . .
. . . . . . . . . . . . . . 1 . 1 . . . . . . . . . . . . . .
. . . . . . . . . . . . . . 1 1 1 . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

20 31 0 1
1
1 3 15
5 5 5 5 5 6 0 0 0 0 0 7 6 6 5 4 4 4 4 4 6 7 0 7 7 7 6 0 0 2 1 1 1 0 1 2 4 4 4 4 4 3 2 2 1 0 0 0 0 0 2 3 3 3 3 3 4 4
```

```
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . 2 2 2 . . . . . . . .
. . . . . 1 1 . 1 1 . . . . . . . . . . . . . . . . . . 2 . . . 2 . . . . . . .
. . . . 1 . . . . 1 . . . . . . . . . . . . . . . . . 2 . . . . . 2 . . . . . .
. . . . . 1 1 . . 1 . . . . . . 1 . . . . . . . . . 2 . . . . . . . 2 . . . . .
. . . . . . . 1 . 1 . . . . . . 1 . . . . . . . . 2 . . . . . . . . . 2 . . . .
. . . . 1 1 1 . . 1 . . . . . . 1 . . . . . . . . 2 . . . . . . . . . 2 . . . .
. . . . 1 . . . . 1 . . . . . 1 . 1 . . . . . . . 2 . . . . . . . . . 2 . . . .
. . . . 1 . . . . 1 . . . . 1 . . . 1 . . . . . . . 2 . . . . . . . 2 . . . . .
. . . . 1 . . . . 1 . . . 1 . . . . 1 . . . . . . . . 2 2 2 2 2 2 2 . . . . . .
. . . . 1 . . . . . 1 . 1 . . . . . . 1 . . . . . . . . . . . . . . . . . . . .
. . . . 1 . . . . . . 1 . . . . . . . . 1 . . 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 . .
. . . . 1 . . . . . . . . . . . . . . . 1 . . . 3 . . . . . . . . . . . 3 . . .
. . . . 1 . . . . . . . . . . . . . . . 1 . . . . 3 . . . . . . . . 3 3 . . . .
. . . . . 1 . . . . . . . . . . . . . 1 1 . . . 3 . . 3 . . . . . 3 . . . . . .
. . . . . . 1 . . . . . 1 1 1 . . . 1 . . . . 3 . . 3 . 3 . . . 3 . . . . . . .
. . . . . . . 1 1 . . 1 . . . 1 1 . 1 . . . . 3 . 3 3 . . 3 3 3 . . . . . . . .
. . . . . . . . . 1 1 . . . . . . 1 . 1 . . . 3 3 . . . . . . . . . . . . . . .
. . . . . . . . . 1 . . . . . . . . 1 . . . . . 3 . . . . . . . . . . . . . . .

20 40 0 3
3
1 3 8
5 4 5 7 0 7 5 4 4 6 6 6 6 6 6 6 7 7 7 0 7 6 1 1 1 0 0 7 0 7 7 1 3 2 1 0 2 2 2 3 3 2 3 3 2 2 6 6 5 5 5 5 5 3 3 2 2 2 2 2 2 2 4 3
2 3 30
5 5 5 5 6 6 7 7 0 0 0 0 0 0 1 1 2 2 3 3 3 3 4 4
3 13 24
7 7 5 5 6 6 7 2 1 0 2 1 7 7 0 0 1 1 1 0 1 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```