1 Option:

I choose Option 1, didn't use "for" or "while" at all.

2 Cases:

I believe I handled every case (F, R, C, D and N). However, I might have missed some bugs.

3 Code:

I'm writing this method before the 2nd one was announced. I'm still not sure about the code part

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;
public class OCB2017400048 {
	public static void main(String[]args) throws FileNotFoundException {
		Scanner file = new Scanner(new File("input.txt"));
		String size=file.next();
		int row=Integer.parseInt(size.substring(0,size.indexOf("x")));
		int column=Integer.parseInt(size.substring(size.indexOf("x")+1));
		String[][] numbers=new String[row][column];
		numberReader(numbers, 0, 0, row-1, column-1, file);
		file.close();
		numberPrinter(numbers, 0, 0, row-1, column-1);
		numberProcesser(numbers, 0, 0, row-1, column-1);
		System.out.println();
		numberPrinter(numbers, 0, 0, row-1, column-1);
		/*
		 I'm sorry that the code is so messy but due to the finals, I couldnt organise it. Some methods return Strings, others return int, some are just void etc. but hey it
works!
		 */
	}
		public static void numberReader(String[][] numbers, int row, int column, int realRow, int realColumn, Scanner file) {
			//this method reads the input.txt file and saves it into numbers[row][column] String array.
			if (column==realColumn) {
				if(row==realRow) numbers[row][column]=file.next();
				else {
					numbers[row][column]=file.next();
					numberReader(numbers, row+1, 0, realRow, realColumn, file);
				}
			}
			else {
				numbers[row][column]=file.next();
				numberReader(numbers, row, column+1, realRow, realColumn, file);
			}
		}
		public static void numberPrinter(String[][] numbers, int row, int column, int realRow, int realColumn) {
			//this method prints the entire number board.
			if(column==realColumn) {
				if(row==realRow) System.out.println(numbers[row][column].substring(1));
				else {
					System.out.println(numbers[row][column].substring(1));
					numberPrinter(numbers, row+1, 0, realRow, realColumn);
				}
			}
			else {
				System.out.print(numbers[row][column].substring(1)+" ");
				numberPrinter(numbers, row, column+1, realRow, realColumn);
			}

		}
		public static void numberProcesser(String[][] numbers,int row, int column, int realRow, int realColumn) {
			//this method proceeds if a number needs to be changed, repeats untill the very last number.
			int[] tempColumn=new int[realRow+1];
```

```
		int sum=0;
		int[] counter=new int[1];
		if(column==realColumn) {
			if(row==realRow) {
				//PROCESS
				if(numbers[row][column].startsWith("R")) numbers[row][column]="R"+String.valueOf(findBiggest(numbers, row, 0,
realColumn));
				if(numbers[row][column].startsWith("C")) {

					takeColumn(numbers, tempColumn, column, 0, realRow);

					numbers[row][column]="C"+String.valueOf(process_C(tempColumn, 0, realRow));
				}
				if(numbers[row][column].startsWith("D")) numbers[row][column]="D"+String.valueOf(process_D(numbers, 0, 0, realRow,
realColumn, row+column, row-column, counter)/counter[0]);
				if(numbers[row][column].startsWith("N")) process_N(numbers, row, column);
				//PROCESS ENDS
			}
			else {
				//PROCESS
				if(numbers[row][column].startsWith("R")) numbers[row][column]="R"+String.valueOf(findBiggest(numbers, row, 0,
realColumn));
				if(numbers[row][column].startsWith("C")) {

					takeColumn(numbers, tempColumn, column, 0, realRow);

					numbers[row][column]="C"+String.valueOf(process_C(tempColumn, 0, realRow));
				}
				if(numbers[row][column].startsWith("D")) numbers[row][column]="D"+String.valueOf(process_D(numbers, 0, 0, realRow,
realColumn, row+column, row-column, counter)/counter[0]);
				if(numbers[row][column].startsWith("N")) process_N(numbers, row, column);
				//PROCESS ENDS
				numberProcesser(numbers, row+1, 0, realRow, realColumn);
			}
		}
		else {
			//PROCESS
			if(numbers[row][column].startsWith("R")) numbers[row][column]="R"+String.valueOf(findBiggest(numbers, row, 0, realColumn));
			if(numbers[row][column].startsWith("C")) {

				takeColumn(numbers, tempColumn, column, 0, realRow);

				numbers[row][column]="C"+String.valueOf(process_C(tempColumn, 0, realRow));
			}
			if(numbers[row][column].startsWith("D")) numbers[row][column]="D"+String.valueOf(process_D(numbers, 0, 0, realRow, realColumn,
row+column, row-column, counter)/counter[0]);
			if(numbers[row][column].startsWith("N")) process_N(numbers, row, column);
			//PROCESS ENDS
			numberProcesser(numbers, row, column+1, realRow, realColumn);
		}
	}

	public static String process_R(String[][] numbers, int row, int realColumn) {
		//this and the findBiggest methods are used to change "R" numbers.
		return "R"+findBiggest(numbers, row, 0, realColumn);
	}
	public static int findBiggest(String[][] numbers, int row, int column, int realColumn) {
		//finds the biggest number in the row.
		if(column==realColumn) return Integer.parseInt(numbers[row][column].substring(1));
		else return Math.max(Integer.parseInt(numbers[row][column].substring(1)), findBiggest(numbers, row, column+1, realColumn));
	}

	public static void takeColumn(String[][] numbers, int[]tempColumn, int column, int row, int realRow) {
		//takes the necessary column into an array.
		if(row==realRow) {
			tempColumn[row]=Integer.parseInt(numbers[row][column].substring(1));
		}
		else {
			tempColumn[row]=Integer.parseInt(numbers[row][column].substring(1));
			takeColumn(numbers, tempColumn, column, row+1, realRow);
		}
```

```
	}
	public static int process_C(int[] tempColumn, int row, int realRow) {
		//this and the takeColumn methods are used to change "C" numbers.
		Arrays.sort(tempColumn);

		return tempColumn[(realRow+2)/2-1];
	}

	public static int process_D(String[][] numbers, int row, int column, int realRow, int realColumn, int whole, int difference, int[] counter) {
		//if "row+column or row-column" are equal to the original, it proceeds
		if (column==realColumn) {
			if(row==realRow) {
				if(row+column==whole||row-column==difference) {
					counter[0]++;
					return Integer.parseInt(numbers[row][column].substring(1));
				}
				else return 0;
			}
			else {
				if(row+column==whole||row-column==difference) {
					counter[0]++;
					return Integer.parseInt(numbers[row][column].substring(1))+process_D(numbers, row+1, 0, realRow, realColumn,
whole, difference, counter);
				}
				else return process_D(numbers, row+1, 0, realRow, realColumn, whole, difference, counter);
			}
		}
		else {
			if(row+column==whole||row-column==difference) {
				counter[0]++;
				return Integer.parseInt(numbers[row][column].substring(1))+process_D(numbers, row, column+1, realRow, realColumn, whole,
difference, counter);
			}
			else return process_D(numbers, row, column+1, realRow, realColumn, whole, difference, counter);
		}


	}

	public static void process_N(String[][] numbers, int row, int column) {
		//look 4 ways, if N, change its number to yours.
		try {
			if(numbers[row-1][column].startsWith("N")) numbers[row-1][column]="N"+numbers[row][column].substring(1);
		}
		catch(ArrayIndexOutOfBoundsException e){

		}
		try{
			if(numbers[row+1][column].startsWith("N")) numbers[row+1][column]="N"+numbers[row][column].substring(1);
		}
		catch(ArrayIndexOutOfBoundsException e){

		}
		try{
			if(numbers[row][column-1].startsWith("N")) numbers[row][column-1]="N"+numbers[row][column].substring(1);
		}
		catch(ArrayIndexOutOfBoundsException e){

		}
		try{
			if(numbers[row][column+1].startsWith("N")) numbers[row][column+1]="N"+numbers[row][column].substring(1);
		}
		catch(ArrayIndexOutOfBoundsException e){

		}
		//change yourself from N to F, so that with recrusive you wont be in a loop forever.
		numbers[row][column]="F"+numbers[row][column].substring(1);

		//look 4 ways, if N, recrusive
		try{
			if(numbers[row-1][column].startsWith("N")) process_N(numbers, row-1, column);
```

```
                }
                catch(ArrayIndexOutOfBoundsException e){

                }
                try{
                        if(numbers[row+1][column].startsWith("N")) process_N(numbers, row+1, column);
                }
                catch(ArrayIndexOutOfBoundsException e){

                }
                try{
                        if(numbers[row][column-1].startsWith("N")) process_N(numbers, row, column-1);
                }
                catch(ArrayIndexOutOfBoundsException e){

                }
                try{
                        if(numbers[row][column+1].startsWith("N")) process_N(numbers, row, column+1);
                }
                catch(ArrayIndexOutOfBoundsException e){

                }
                //catch outOfBoundsException and {} it
        }
}
```

4 Examples: