/* Problem 3--Always Look On The Bright Side Of Life
   The way I did it was I located LIFE and BRIGHT and treated the axis
   containing LINE as a linear inequality. I tested which side of the
   inequality BRIGHT was on, and looked for a BRIAN that was on the
   other side. */

import java.io.*;
import java.util.*;

public class prob3 {
    private static Scanner in;
    private static PrintWriter out;
    private static int cs, r, c;
    private static char[][] Grid;

    public static void main (String[] args) throws Exception {
        in = new Scanner (new File ("prob3.in"));
        out = new PrintWriter ("prob3.out");
        cs = 1;
        while (true) {
            r = in.nextInt(); //Get row and col information
            c = in.nextInt();
            if (r==0 && c==0) break;
            in.nextLine();
            ReadIn (); //Get the word grid
            Process (); //Solve the problem
        }
        in.close();
        out.close();
    }

    /* Read in the grid a row at a time */
    public static void ReadIn () throws Exception {
        Grid = new char[r][c];
        for (int i=0; i < r; i++) Grid[i] = in.nextLine().toCharArray();
    }

    /* Search for words and check for the BRIGHT side of LIFE. */
    public static void Process () throws Exception {
        Location[] LIFE = Find ("LIFE"); //Find all instances of LIFE, BRIGHT,
        Location[] BRIGHT = Find ("BRIGHT"); //and BRIGHT. There had better be
        Location[] BRIAN = Find ("BRIAN"); //only one LIFE and only one BRIGHT
        int brightside = side (LIFE[0].getr(),LIFE[0].getc(),
                             LIFE[0].getc()+LIFE[0].getdr(),LIFE[0].getc()+LIFE[0].getdc(),
                             BRIGHT[0].getr(),BRIGHT[0].getc()); //which side of LIFE is
        int i=0; //BRIGHT on, positive or negative?
        for (;i++;) //Go through all the BRIANs until you get one whose side
            if (brightside==side (LIFE[0].getr(),LIFE[0].getc(), //is brightside
                                    LIFE[0].getc()+LIFE[0].getdr(),LIFE[0].getc()+LIFE[0].getdc(),
                                    BRIAN[i].getr(),BRIAN[i].getc()) &&
                brightside==side (LIFE[0].getr(),LIFE[0].getc(),
                                 LIFE[0].getc()+LIFE[0].getdr(),LIFE[0].getc()+LIFE[0].getdc(),
                                 BRIAN[i].getr()+BRIAN[i].getdr()*4,
                                 BRIAN[i].getc()+BRIAN[i].getdc()*4)) break;
        for (int p = 0; p < 5; p++) //Lowercase all the letters of that BRIAN
            Grid[BRIAN[i].getr()+BRIAN[i].getdr()*p]
            [BRIAN[i].getc()+BRIAN[i].getdc()*p] += 32;
        Print (); //Print the grid
    }
}
/* Just prints the new grid */
public static void Print () throws Exception {
    out.printf("Case %d:\n\n",cs++);
    for (int i=0; i < r; i++)
        out.printf("%s\n",new String(Grid[i]));
    out.printf("\n");
}

/* If you have two points (a,b),(c,d), the line going through them is
  given by (b-d)x + (c-a)y +ad-bc = 0.  Some (x,y) might make this
  formula positive and some might make it negative; that's how you
  know which side of the line you're on. */
private static int side (int a, int b, int c, int d, int row, int col)
    throws Exception {
    int comp = (b-d)*row+(c-a)*col + a*d-b*c;
    int side;
    if (comp > 0) side = 1;
    else if (comp < 0) side = -1;
    else side = 0;
    return side;
}

/* Find locates all instances of a word in the grid and stores them in
  an array */
public static Location[] Find (String word) throws Exception {
    Location[] l = new Location[r*c];
    int lct = 0;
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            for (int dr=-1; dr <= 1; dr++) //Loop through all directions
                for (int dc=-1; dc <= 1; dc++) //horizontally vertically
                    if (dr!=0 || dc != 0) //diagonally
                        String thisword="";
                        for (int k=0; k < word.length(); k++)
                            thisword += G(i+k*dr,j+k*dc); //Build word
                        if (thisword.equals(word)) l[lct++] = new Location (i,j,dr,dc);
                        //Did we find it?  Add location and direction to array
    return l;
}

/* Easy way to look in the array.  It returns a space is we go out of
  bounds...thus preventing us from going out of bounds! */
public static char G (int i, int j) throws Exception {
    char letter;
    if (i < 0 || i >=r || j < 0 || j >= c) letter = ' ';
    else letter = Grid[i][j];
    return letter;
}

/* Location stores the first letter of the word [r,c] and the direction
  of the word [dr,dc].  Forward is [1,0].  Backward is [-1,0].
  Northeast = [1,-1], and so forth. */
class Location {
    private int r,c,dr,dc;
    /* Accessors */
    public int getr () throws Exception {return r;}
    public int getc () throws Exception {return c;}
}
public int getDr () throws Exception {return dr;}
public int getDc () throws Exception {return dc;}

/* Constructor */
public Location (int R, int C, int DR, int DC) throws Exception {
r = R; c = C; dr = DR; dc = DC;
}