CS 100 Spring 1998
Assignment 6
"Arrays and Even More Loops"

Due: *at the end of lecture Tuesday March 10. Assignments will not be accepted late*

This time there are essentially two questions. The first should be done be done in the same style as those of the previous assignment, using static functions as appropriate inside main. The second will use classes which in main will be instantiated. To make the organisation of this easier, you will be directed to add code to files to be downloaded from the web site. The question itself gives an overview of how that program works, together with hints on constructing the missing code. In all these questions, we ask you to test your program with suggested data (and you should turn in your code together with the output with this data as a minimum), however, we recommend that you also run your programs with other data of your own choosing. You are encouraged to work with one partner.

1. Two exercises manipulating 1–dimensional arrays.

(i) Write a program which will take a given array $A[\ ]$ of `doubles`, and replace each entry by the average up to (and including) that term in the array. Use as little extra storage as you can to do this. As an example, if

$$A[\ ] = \{1\,,\ 3\,,\ 11\,,\ 13\,,\ 5\,,\ 21\},$$

then $A[\ ]$ would become

$$A[\ ] = \{1\,,\ 2\,,\ 5\,,\ 7.5\,,\ 6.6\,,\ 9\}.$$

You should test your program by applying it to the above example, and also to the array comprising all the prime numbers from 3 up to 97 inclusive. (It's up to you whether merely to write in the values of these primes, or to write code to generate them – whatever amuses you!)

(ii) Write a program which will take a given array $B[\ ]$ of `integers`, and reverse the order of its terms. Be careful to check that your program works if $B[\ ]$ should have an odd number of terms or an even number of terms. Again, use as little extra storage as you can to do this. As an example, if

$$B[\ ] = \{1\,,\ 3\,,\ 11\,,\ 13\,,\ 5\,,\ 21\},$$

then $B[\ ]$ would become

$$B[\ ] = \{21\,,\ 5\,,\ 13\,,\ 11\,,\ 3\,,\ 1\}.$$

You should test your program by applying it to the above example, and also to the array comprising all the prime numbers from 17 up to 97 inclusive.

2. This is an extension of last time's Lateasean land agent problem. The objective is to compute the area of any region bounded by the kinds of fences we had on Assignment 5. To keep things simple, we'll restrict ourselves to triangular regions. This will also be a class-based exercise in inheritance, but to ease your sanity, you will be adding code where marked in a file you can download from the course web site. (By the time you've finished this problem, you should ensure that you understand how this whole program works.)

In 1899, Pick (an ancestor of the First Surveyor) discovered that the area of any of these Lateasean regions is always exactly equal to

$$\frac{1}{2}(\texttt{boundary}) + \texttt{inside} - 1,$$

where `boundary` is the number of calumin stakes *on* the boundary, and `inside` is the number of stakes genuinely *inside* the boundary. You already have a function which will find the number of stakes on a straight fence, so you will find yourself using this to count the number of stakes on a triangular boundary. You will have to write from scratch a function to count the inside points, but we will give you some major hints for this shortly.

There will be three points P and Q and R, given by pairs of integers `x1,y1` and `x2,y2` and `x3,y3`. The overall organisation of the program has a 'parent' class `Grid` and a 'child' class `GridTriangle`.

The class `Grid` has fields for P and Q (also fields n for size of square grid, `draw` explained shortly, x and y for the location of the upper left corner of the picture, and `spacing` for the spacing of the dots — all these are used by the program internally, you can use these, but don't change them), two explicit constructors for initialising any instance of `Grid`, a function `fence` (which you've essentially already written!), a boolean `draw` which is used to check if the picture will be too big to be worth attempting to draw (this is written and used for you), a function `drawGrid` which draws a square grid of 'stakes' (you'll write a pair of nested loops together with `fillOval` as indicated in the downloaded file) and also draws the fence from P to Q (this bit is written for you).

The class `GridTriangle` inherits P, Q (and the other internal fields listed above), `fence`, and `drawGrid` from `Grid`, and additionally has a field of its own for R (as the pair of integers `(x3,y3)`. It also has its own constructor for initialising any instance of `GridTriangle` exploiting its parent's constructor (courtesy of `super`). It has some functions of its own, namely a function `boundary` (which you will define using three calls to `fence`) which uses the inherited `fence` function to return the number of stakes on the boundary (be careful not to double-count any stakes), a function `inside` to return the number of stakes inside the boundary (hints follow), a function `drawGridTriangle` which uses the inherited `drawGrid` to draw the grid and then has its own calls to `drawLine` to draw the three fences of its boundary (this is written for you), and finally a function `area` to return the area.

The way all this might be used is

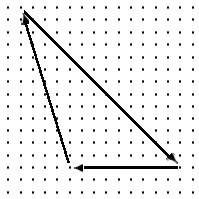
```
Grid Sausage = new Grid(1,2,11,20);
int stakes = Sausage.fence(1,2,11,20);
```

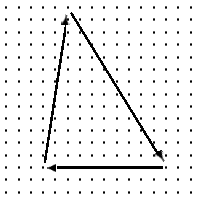
which will draw a grid together with Sausage's fence from $(1, 2)$ to $(11, 20)$, provided the requested grid isn't too big to draw, and then make `stakes` take the value of the number of stakes along the fence. More usefully,

```
GridTriangle Rhubarb = new GridTriangle(1,2,11,20,7,16);
int entrails = Rhubarb.inside();
int skin = Rhubarb.boundary();
double bigness = Rhubarb.area();
```

which will draw a grid together with Rhubarb's triangle, and then makes `entrails`, `skin` and `bigness` take the values of the number of inside stakes, the number of boundary stakes, and the area of Rhubarb, respectively.

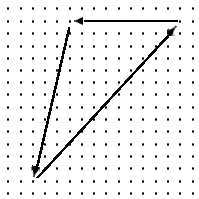
Now for some hints for the `inside` function. Consider a triangle whose base is horizontal.

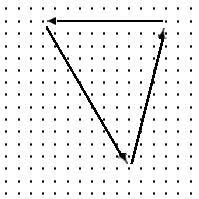




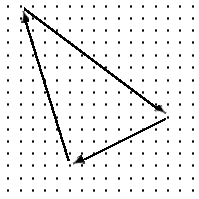
Let the max vertex y–values be `bot` and the min y–value be `top` (Java coordinates!). Since there won't be any inside points with `bot` or `top` as their y–values, have an outside loop counting from `y = top + 1` to `bot - 1`. Now for each `y`, have an inside loop testing how many `x`'s on that horizontal line lie between the two sloping lines. To do this, let the min vertex x–value be `left` and the max x–value be `right`, then as `x` moves from `left - 1` to `right + 1` (for each `y`), check if the slope of the line from the top point to (x, y) lies between the slopes of the left and right sides of the triangle. (Note that `Math.max(a,b)` and `Math.min(a,b)` each take only two arguments.)

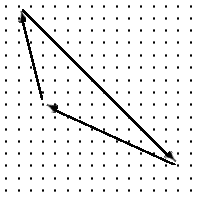
It shouldn't be hard for you to modify this to deal with a triangle whose *top* is horizontal.





All that remains is to handle the more general picture.





You already now have routines to do the special cases we looked at, so first test to see if `y1`, `y2` and `y3` are all different. If two of them are the same, then you have an earlier special case. If they are all different, then let the max be `BOT`, the min be `TOP`, and the other one be `MID`. Imagining a horizontal line drawn with y–value `MID` almost gives a copy of each of our earlier special cases — almost, because although the x–value for the *vertex* with `y = MID` is integer, the x–value for the facing side when `y = MID` is probably not integer. However, this is not a problem for us. Since we did our `x` moving from `left - 1` to `right + 1`, if we still check the slopes as before, we will get a correct count in each 'half' of the original triangle, though to get the boundary line slopes you must use the original triangle's vertices. The only remaining thing is to remember to count (once!) the stakes along the *imaginary* horizontal line you drew to separate the general triangle into the two special cases! (Note that to find the max of three numbers you will have to do something like `Math.max(a , Math.max(b,c))`, and similarly for min.)

Try this with several triangles of your choice, including the triangles $P = (x1, y1) = (2, 1)$, $Q = (x2, y2) = (20, 25)$ and $R = (x3, y3) = (17, 19)$ and the triangle $P = (x1, y1) = (2, 1)$, $Q = (x2, y2) = (2106, 2587)$ and $R = (x3, y3) = (466, 1257)$.