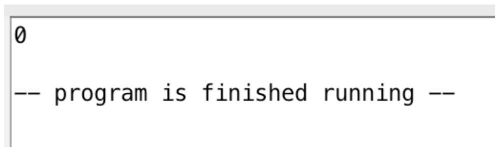
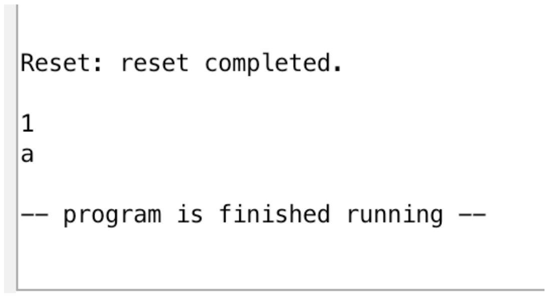
Project 1: Preston Hansen

I/O Demonstrations:

For the base case of parts 1 and 2 (n = 0,1), a 1x1 matrix is printed, containing just one character.

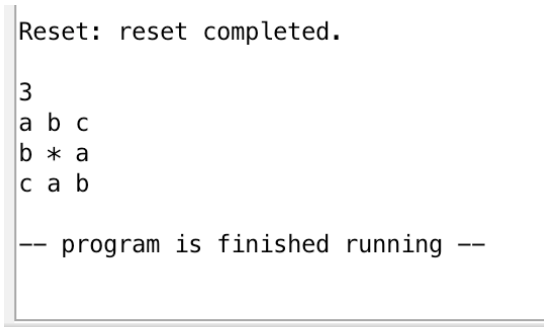


```
0

-- program is finished running --
```

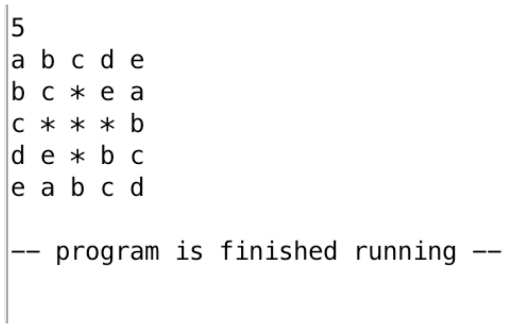


```
Reset: reset completed.

1
a

-- program is finished running --
```

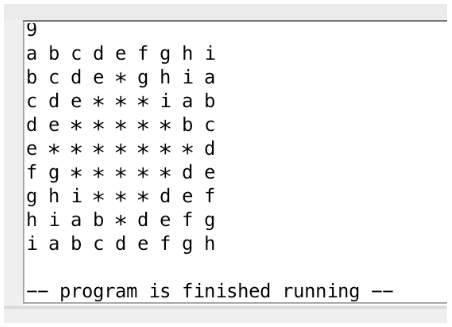
For higher n, a nxn matrix is printed with an n-2 width, n-2 height diamond in the middle.



```
Reset: reset completed.

3
a b c
b * a
c a b

-- program is finished running --
```

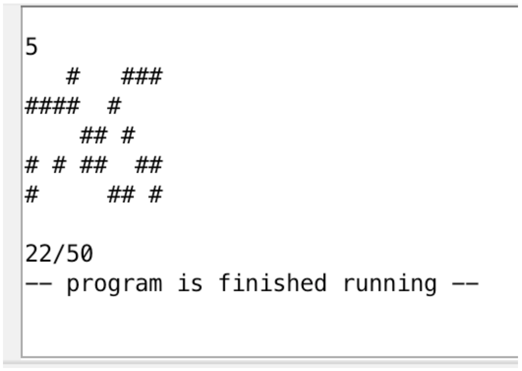


```
5
a b c d e
b c * e a
c * * * b
d e * b c
e a b c d

-- program is finished running --
```

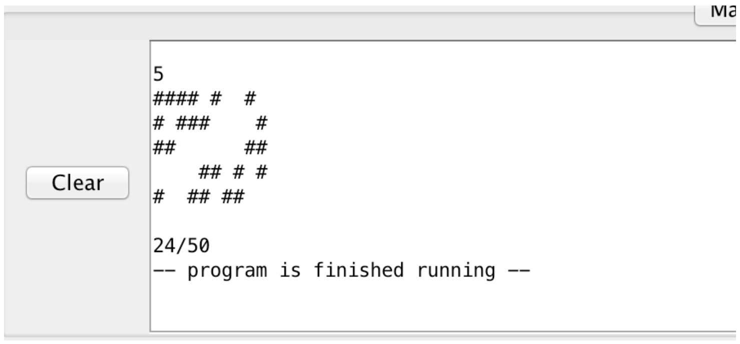


```
9
a b c d e f g h i
b c d e * g h i a
c d e * * * i a b
d e * * * * * b c
e * * * * * * * d
f g * * * * * d e
g h i * * * d e f
h i a b * d e f g
i a b c d e f g h

-- program is finished running --
```

For Part 3a, a n x 2n matrix is printed with approximately half of the spaces filled with pound symbols, randomly.



```
5
   #   ###
####  #
    ## #
# # ##  ##
#     ## #

22/50
-- program is finished running --
```



```
5
#### #  #
# ###    #
##      ##
    ## # #
#  ## ##

24/50
-- program is finished running --
```

Program Features:

I completed levels 1, 2, and part a of level 3. All of the specified features work as intended. For levels 1 and 2, an NxN matrix is printed with a diamond of asterisks height (N-2) and width (N-2). The “wrap back” of characters functions as well. For 3a, a Nx2N matrix is printed with approximately 50% of the spaces filled with pound symbols at random, and the total number out of the possible total displayed.

Implementation Details:

a) **Part 1**: The looping structure used was to store a counter for the row, a counter for the column, the user input (N), and the maximum character printed in the matrix (‘a’ + N). At the beginning of each loop, ‘a’ is loaded into a register, and then summed with the row counter (initially zero) to determine the first character of a row. That is, row 1 begins with a, 2 with b, etc. After the first character is printed, a space is printed and the counters for column and character are incremented. If the character counter equals ‘a’ + N, then the character is reset to ‘a’. If the column less than N, the program branches back to the character printing label. If the column equals N, a newline character is printed, the column counter is reset to zero, and the row counter is incremented. Then, the program branches to the beginning of the loop (which resets the character to ‘a’, and sums ‘a’ and the row number). Finally, if the row number reaches N, the program ends.

b) **Part 1 (cont’d)**: The character which is printed at any given location is determined as described above. That is, starting at ‘a’ and increasing to ‘a’ + N, characters are printed. If the character reaches ‘a’ + N, it is reset to ‘a’. At the beginning of each row, the character is ‘a’ + row number (0-N).

c) **Part 2**: The number of stars was determined by noting location relative to the middle row of the matrix, calculated by (N/2) + 1. For row 1, the number of stars is -1 (interpreted to mean 0). This increases by two each row (-1, 1, 3, etc) until the middle row is reached. After that point, the number of stars is (N-2, N-4, … , 3, 1, -1). To implement this, when printing the stars, if the current row was before or equal to the middle row, the program would branch to a label which added 4 to the number of stars, then would immediately subtract 2 stars (the net effect being +2 stars). If the row was past the middle row, the program simply branches to the label which subtracts 2 stars from the total. To determine location of stars, I noted that the location was just the middle column, plus or minus (number of stars / 2). This fact was used to create a range of columns for which a star would be printed in a given row. If the potential location of a star was outside the range, the program would branch back to the character printing loop like normal.

d) **Part 3a)**: To ensure a reasonably random distribution, I simply sampled a random integer from 0-1000 (using the $v0 value 42 and $a1 value of 1001). For values of 0-499, a space was printed. For values of 500-1000, a # was printed. The result is that the program prints out a different pattern every time (the odds of receiving the same pattern twice in a row are $0.5^{(2*N*N)}$, or 1 in 1125899906842624 when N = 5).