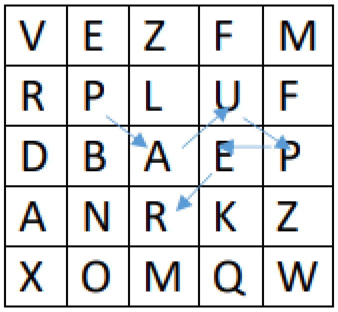
Make Up Set
Total Points: 30

Three problems must be implemented for full credit. The required (starred) problem marked in the set must be implemented by everyone. See the 2436 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points each)

Section One- Choose two problems.

1. The board game Boggle consists of a square board of letters. Players try to form words out of adjacent letters. Here is an example 5x5 board, with the letters used to form the word "`PAUPER`":



V	E	Z	F	M
R	P	L	U	F
D	B	A	E	P
A	N	R	K	Z
X	O	M	Q	W

Words can start anywhere and can be formed by connecting letters adjacent in any direction, including diagonally. Words cannot use the same letter in the same location more than once. For example, the word "`FLUFF`" is not able to be spelled on the above board, because one must use one of the two `F`'s in the board twice. Write a C++ program that takes a sample board, and a set of sample words, and determines which ones can or cannot be spelled on the board.

Input will be from a data file. Each input instance will begin with a positive integer `N` between `[1,8]` denoting the size of the board with `n = 0` denoting the end of input. There will then follow `N` lines of `N` capital letters separated by spaces. There will then be an integer `m` in the range `[1,100]` denoting the number of words to verify on each board. There will follow `m` strings of words, one per line. Each string will be 7 characters or less denoting the words to check for on the board. For each input instance `i`, output to the screen: "`Board i`" then for each word `j` on that board, output "`Word j: YES`" if it is possible to spell the word on the board, or "`Word j: NO`" if it is not. Separate each board with a blank line. The program must use recursion for full credit. Let the user enter the file name from the keyboard. Refer to the sample output below.

Sample File:

```
 5
 V E Z F M
 R P L U F
 D B A E P
 A N R K Z
 X O M Q W
 4
```

Sample Run:

```
Enter file name: boggle.txt

Board 1
Word 1: YES
Word 2: NO
Word 3: YES
Word 4: NO
```

```
PAUPER
FLUFF                                Board 2
BRAKE                                Word 1: YES
QUIZ                                 Word 2: NO
4                                    Word 3: NO
P O W A                              Word 4: YES
E S I T
R S L U
E I N T
4
POSERS
SILENT
SPORE
SLIT
0
```

Name the program: `BoggleCheckXX.cpp`, where `XX` are your initials.

2. Write a C++ program to count blobs in a grid. The user will enter the row and column grid size from the keyboard in the range `[1,50]` on separate lines. To generate the blob let the user enter the percentage probability of blob characters as a percentage between `[0,100]`. Error check inputs. The program will then randomly generate the data in a two-dimensional grid of cells, each of which may be empty (value `-`) or filled (value `X`). A blob consists of one or more `X`'s connected horizontally, vertically, or diagonally. Output to the screen the generated blob and total number of blobs found in the grid. The program must use recursion for full credit. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter row size [1,50]: 10
Enter column size [1,50]: 40

Percentage probability [0,100]: 10

---------X------------------X-----------
--XX----------X-------------------------
----XX---------X---------------------X--
---X-X-------------X------X-------------
--------------------------------X-----XX
------X-X-------X-------------X-X---X---
X------------------------------X--------
------------------------X--------X-X----
------X------X----X---XX-----------X----
---------------X---------X--------------

There are 22 blobs.

Run Again (Y/N): Y

Enter row size [1,50]: 10
Enter column size [1,50]: 40
```

```
Percentage probability [0,100]: 80

-XXXXX-XXXXXXXX-XXX--XXXX-XX--XXXXXXXXX-
XXXXXXXXXXXXXXXXXXXX-XXXXXX-XXXX-XXXXXXX
X-XX--X---XXXXXXX--XXXXXXXXX-XXXX-XXXXXX
XXXXXXXXXXXXXXXXXXXXX-XXXXXXXX-XX--XXXX-
XX--XX-XX-XX-X-XXXXX-XXX-XXX-XXXXXX-XXXX
XXXXX-XXXX-X-XX--XXXXX-X-X-XXX-XXX--X-XX
XX-XXXXX-XXX-XXXXXXXXXXXXXX-XX-XX--X-X-X
XXXXXXXXXX-XXXXXXXXXXXXXXXXXX-XXX-XXXXXX
XXXXXXXXXXXXXXXXXX-X-XXX-XX-XXX-XX-XXXXX
-X-XXXXXXXXXXXXXXXXXXXXXXXXX-XX----XXXXX

There is 1 blob.

Enter row size [1,50]: 5
Enter column size [1,50]: 5

Run Again (Y/N): Y

Percentage probability [0,100]: 50

-XX--
XXX--
XX--X
--XXX
X-X-X

There are 2 blobs.

Run Again (Y/N): Y

Enter row size [1,50]: 6
Enter column size [1,50]: 7

Percentage probability [0,100]: 30

X-----X
------X
-------
-----XX
X-----X
--XX---

There are 5 blobs.

Run Again (Y/N): n
```

Name the program: `BlobGeneratorXX.cpp`, where XX are your initials.

3. Write a C++ program given a person's one-word first name, construct a binary search tree and output the name using five traversals: in-order, pre-order, post order, level order, and reverse order. A reverse order traversal is an in-order traversal backwards. Allow duplicate letters and slide those to the left as they reach a matching node. Input from the keyboard a one-word first name in all caps greater than two letters long. Assume proper input. Output to the screen, the original name and the name using each of the five traversals listed above on separate lines with one space between each character and properly labeled. The program must use a tree data structure. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

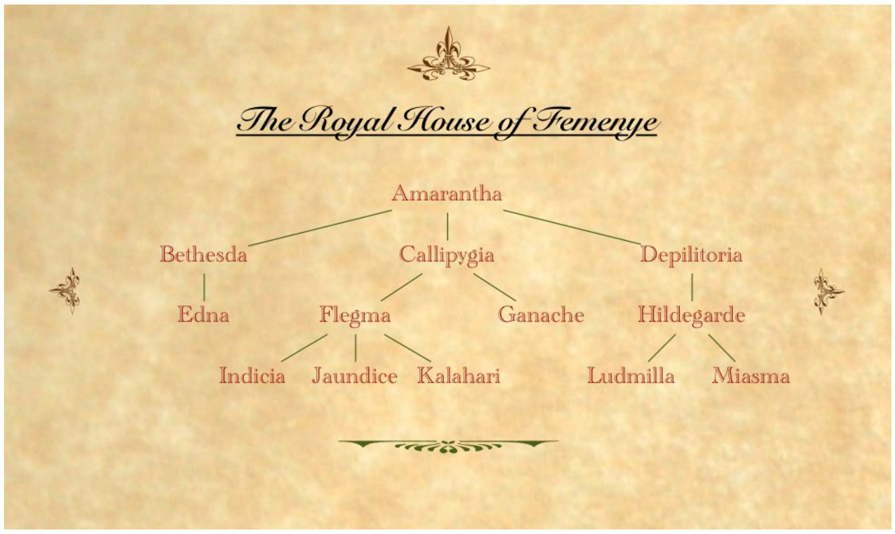
```
Enter a last name: JAMES

Traversals:

In Order:           A E J M S
Pre Order:          J A E M S
Post Order:          E A S M J
Level Order:        J A M E S
Reverse Order:      S M J E A

Run Again (Y/N): Y

Enter a last name: EKATERINA

Traversals:

In Order:           A A E E I K N R T
Pre Order:          E A A E K I T R N
Post Order:          A E A I N R T K E
Level Order:        E A K A E I T R N
Reverse Order:      T R N K I E E A A

Run Again (Y/N): n
```

Name the program: `NameTraversalsXX.cpp`, where XX are your initials.

Required Problem- Comprehensive.

4 (**). Write a C++ program to determine the royal order of succession for a fictious kingdom. The Royal Family of Femenye needs help to determine the proper order of succession in the monarchy before the impending death of the current queen. The monarchy of Femenye is also a matriarchy. Thus, the rules of succession concern only the female half of the royal family tree. In particular, the rules state that the eldest daughter of the current queen will succeed her, followed by her next eldest sister, and so on through the daughters of the current queen. If none of the daughters of the current queen are available to serve, the succession proceeds to the daughters of the "first daughter", in order from oldest to youngest, then the daughters of the next eldest daughter of the current queen, etc. In the picture

below, the daughters of a given female royal are shown beneath her and are connected to her by lines, in order from eldest to youngest, proceeding from left to right.



Therefore, the order of succession would be as follows including the Queen herself for completeness:

```
Amarantha   (Current queen)
Bethesda    (Her eldest daughter)
Callipygia
Depilitoria
Edna
Flegma
Ganache
Hildegarde
Indicia
Jaundice
Kalahari
Ludmilla
Miasma
```

Note that the succession here just happens to follow in alphabetical order, this is not true in general but was done in this example to make it easier to follow.

Input will be from a datafile. The file will be organized line by line, with no empty lines. The first line will always be just the name of the current queen and the date of her birth, an integer year. All subsequent lines will hold two names and a date as an integer. The first name is the name of a mother and the second the name of a daughter, with the year being the year the daughter was born. A mother's name will always appear on an earlier line in the file, or on the first line as the queen, before it shows up on a line with any of her daughters. Assume all names will be a single word, with no hyphens or other punctuation, and the daughters will always be born at least one year apart. However, daughters of a given mother may appear out of order, and the listing of daughters of a given mother may be interrupted by the listing of several other mother-daughter pairs. Output to the screen a title followed by the female order of succession, including the current queen, based on the datafile given. Let the user enter the file name from the keyboard. Refer to the sample output below.

Sample File:

```
Amarantha                       1792
Amarantha       Depilitoria     1815
Depilitoria     Hildegarde      1831
Amarantha       Callipygia      1812
Hildegarde      Ludmilla        1856
Callipygia      Ganache         1831
Amarantha       Bethesda        1810
Hildegarde      Miasma          1859
Callipygia      Flegma          1828
Flegma          Kalahari        1860
Bethesda        Edna            1830
Flegma          Indicia         1848
Flegma          Jaundice        1850
```

Sample Run:

```
Enter file name: royals.txt

Succession Order:

Amarantha
Bethesda
Callipygia
Depilitoria
Edna
Flegma
Ganache
Hildegarde
Indicia
Jaundice
Kalahari
Ludmilla
Miasma
```

Name the program: `RoyalOrderXX.cpp`, where `XX` are your initials.

Extra Credit: Implement the following problem. See the 2436 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points)

Write a C++ program to determine the accessibility of any node to all other nodes within a graph. It is possible to analyze the network once a shortest path matrix has been created. Two network analysis measures for centrality are: degree centrality (how the nodes directly connect with each other) and closeness centrality (how close they are to each other). For example:

```
A - B  -  C
     \   /
       D
```

	A	B	C	D
A	0	1	2	2
B	1	0	1	1
C	2	1	0	1
D	2	1	1	0

	1	2
A	1	2
B	3	0
C	2	1
D	2	1

Given the graph on the left, the connection matrix in the center shows values that represent how many steps it takes to get to each node from any other node. The longest path in this situation is 2, and so create a two-column "length tally" matrix to represent this situation, like the one on the right. This shows that the A node has one 1-link connection, and two 2-link connections. The B node has three 1-link connections, and C and D each have two 1-link and one 2- link connections. By looking at the graph above, one can intuitively judge that the A node seems to be the most isolated, and that B is clearly the most centralized. To measure the measure the degree centrality, calculate the following:

- Divide the number of 1-link connections for each node by one less than the number of nodes.

Using the length tally matrix shown above, the A node calculation would be 1/3, or 0.33. The B node would be 3/3, or 1.00. The C and D nodes both would be 2/3, or 0.67. Clearly the node with the least degree is A, and the one with the most degree was B. The closeness centrality idea uses the following formula. For each node:

- Find the sum of the number of 1-link connections times 1, 2-link connections times 2, 3-links * 3, etc.
- Divide that sum into the value representing one less than the total number of nodes in the graph.

For node A, the "closeness" calculation would be 3 divided by (1*1 + 2*2), or 3/5, or 0.60. For node B it would be 3 / (3*1 + 2*0), or 3/3, which is 1.00. Nodes C and D both have a "closeness" result of 0.67. This "closeness" measure at first seems to show the same result as the "degree", in that the A node appears to be the most remote node, with the least "closeness" value, and that B is the most central node, with the greatest "closeness" value. Another example:

```
A - B - C - D - E - F - G - H
```

	A	B	C	D	E	F	G	H
A	0	1	2	3	4	5	6	7
B	1	0	1	2	3	4	5	6
C	2	1	0	1	2	3	4	5
D	3	2	1	0	1	2	3	4
E	4	3	2	1	0	1	2	3
F	5	4	3	2	1	0	1	2
G	6	5	4	3	2	1	0	1
H	7	6	5	4	3	2	1	0

	1	2	3	4	5	6	7
A	1	1	1	1	1	1	1
B	2	1	1	1	1	1	0
C	2	2	1	1	1	0	0
D	2	2	2	1	0	0	0
E	2	2	2	1	0	0	0
F	2	2	1	1	1	0	0
G	2	1	1	1	1	1	0
H	1	1	1	1	1	1	1

The example above is essentially a single linked chain of nodes. The connection matrix is shown in the center, and the length tally matrix on the right. Here, there is a six-way tie (BCDEFG) for the nodes with the greatest "degree" (0.29) of connection, however, only two nodes (D and E) share the greatest "closeness" (0.44) measure. The two outermost nodes, A and H, both measure the least in "degree" (0.14) and "closeness" (0.25).

Input will be several data sets from a data file, each consisting of an initial value N, representing the number of nodes in the graph, followed on the next N lines by an NxN matrix of integer values, representing the connection matrix for the graph as described above, with single space separation. Assume the node labels for each graph will always begin with upper-case A and proceed in alpha order for as many nodes as are represented by the graph. For each labeled data set, output to the screen four values, labeled, formatted, and aligned exactly as shown below. List all nodes in alpha order that match each measure, and each output, except for the final set, should be separated by a blank line. Let the user enter the file name from the keyboard. Refer to the sample output below.

Sample File

```
4
0 1 2 2
1 0 1 1
2 1 0 1
2 1 1 0
8
0 1 2 3 4 5 6 7
1 0 1 2 3 4 5 6
2 1 0 1 2 3 4 5
3 2 1 0 1 2 3 4
4 3 2 1 0 1 2 3
5 4 3 2 1 0 1 2
6 5 4 3 2 1 0 1
7 6 5 4 3 2 1 0
6
0 1 1 2 2 2
1 0 2 1 1 1
1 2 0 3 3 1
2 1 3 0 1 2
2 1 3 1 0 2
2 1 1 2 2 0
```

Sample Run:

```
Enter file name: measure.txt

Graph 1:
Least degree 0.33 A
Greatest degree 1.00 B
Least closeness 0.60 A
Greatest closeness 1.00 B

Graph 2:
Least degree 0.14 AH
Greatest degree 0.29 BCDEFG
Least closeness 0.25 AH
Greatest closeness 0.44 DE

Graph 3:
Least degree 0.40 ACDEF
Greatest degree 0.80 B
Least closeness 0.50 C
Greatest closeness 0.83 B
```

Name the program: `GraphMeasureXX.cpp`, where XX are your initials.