Program Set #3
Total Points: 30

Three problems must be implemented for full credit. Each section will state how many problems to implement within it. The required (starred) problem marked in the set must be implemented by everyone. See 2436 Grading Guide Sheet for additional grading/submission information. Partial credit will be given. (10 points each)

Section One- Lists/Linked Lists. Choose one problem.

1. Many people have used pulling petals off roses and tossing them in the air to determine if someone loves them or not. Your sister wants to use roses to determine which one of several suitors she will let take her to the prom. A linked list can be used to represent the prom date suitors. Write a program to simulate her decision and determine her prom date. The simulation will work as follows:

- She has many roses, and she knows how many petals are on each rose.
- The suitors are numbered from 1 to n, where n is the number of suitors.
- She will select one rose and give one petal to suitor #1 and continuing successively through suitors until she runs out of petals. When she reaches the end of the list, she begins over with the beginning of the list.
- The person who gets the last petal is eliminated from consideration.
- She gets another rose and continues the process beginning with the suitor after the one that was eliminated.
- She continues this process until there is only one suitor left – he is her date for the prom.

Input will come from a text file. The first line will contain a single integer n that indicates the number of test cases to be played. For each test case, there are 2 lines. The first line is the list of the first names of the suitors, separated by a comma and a space. The second line has the number of petals on each of the roses separated by a space. The first rose will be used to find the first suitor to be eliminated, the second rose will be used to find the second person to be eliminated, etc. Note: There is one less rose than suitors. Output to the screen, each test case labeled, then display the order of the persons eliminated as `LOSER name`, one per line, with the final line naming the winner as `WINNER name`. Place whitespace between test cases. Let the user input the file name from the keyboard. The program must use a linked list. Refer to the sample output below.

Sample File:

```
1
ALAN, BRYAN, CHAD, DUKE, ERIC, FRED
100 234 564 234 1231
```

Sample Run:

```
Enter file name: dates.txt

Case 1:
LOSER DUKE
LOSER BRYAN
LOSER ALAN
LOSER FRED
LOSER CHAD
WINNER ERIC
```

Name the program: `RosePetalsXX.java` or `RosePetalsXX.cpp`, where XX are your initials.

2. Write a program that solves a classic computer science problem known as the stable marriage problem. The program deals with a group of men and a group of women. The program tries to pair them up to generate as many stable marriages as possible. A set of marriages is unstable if you cannot find a man and a woman who would rather be married to each other than to their current spouses (in which case the two would be inclined to divorce their spouses and marry each other. The input file for the program will list all the men, one per line, followed by a blank line, followed by all the women one per line. The men and women ae numbered according to their positions in the input file (the first man is #1, the second man #2, and so on; the first woman is #1, the second woman is #2, and so on). Each input line (except for the blank line separating men from women list the person's name, followed by a colon, followed by a list of integers. These integers are the marriage partner preferences of this person. For example:

```
Joe: 10 8 35 9 20 22 33 6 29 7 32 16 18
```

indicates that the person is named "Joe" and that his first choice for marriage is woman #10, his second choice is woman #8, and so on. Any women not listed are considered unacceptable to Joe. Output to the screen, the preferred list of stable marriages for each person. Let the user input the file name from the keyboard. A linked list must be used. Refer to the sample output below.

Sample File

```
Man 1: 4 1 2 3
Man 2: 2 3 1 4
Man 3: 2 4 3 1
Man 4: 3 1 4 2
Woman 1: 4 1 3 2
Woman 2: 1 3 2 4
Woman 3: 1 2 3 4
Woman 4: 4 1 3 2
```

Sample Run:

```
Enter file name: stable.txt

Man 1 and Woman 4
Man 3 and Woman 2
Man 2 and Woman 3
Man 4 and Woman 1
```

Name the program: `StableMarriageXX.java` or `StableMarriageXX.cpp`, where XX are your initials.

Section Two- Recursion/Searching/Sorting. Choose one problem.

3. Write a program to generate a grid with randomly placed blob characters, displays the grid, and counts and reports the number of blobs. The user will enter the size of the grid from the keyboard. Max size of 50 rows and columns. To generate the blob let the user enter the percentage probability of blob characters. This percentage will be between 0 and 100. The data are in a two-dimensional grid of cells, each of which may be empty (value -) or filled (value X). A blob can consist of one or more X's connected horizontally, vertically, or diagonally. The program must use recursion for full credit. Refer to the sample output below.

Sample Runs:

```
Enter row size: 10
Enter column size: 40

Input percentage probability (0 to 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 blob(s).

Enter row size: 10
Enter column size: 40

Input percentage probability (0 to 100): 80

-XXXXX-XXXXXXXX-XXX--XXXX-XX--XXXXXXXXX-
XXXXXXXXXXXXXXXXXXX-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-XXXXXXXXXXXXXXXXXXXXXXXX-XX----XXXXX

There are 1 blob(s).

Enter row size: 5
Enter column size: 5

Input percentage probability (0 to 100): 50

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

There are 2 blob(s).


Enter row size: 6
```

```
Enter column size: 7

Input percentage probability (0 to 100): 30

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

There are 5 blob(s).
```

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

4. The TCC Computer Science Club is sponsoring a high school invitational computer science competition to raise money to pay for a trip to a national programming contest. The invitational will be staged in the format of Texas UIL high school competitions where there is a programming portion and a written exam. The club sponsor, Mr. Paul Koester, wants a program to help with grading the written portion of the contest. Each contestants name, name of their school, school's classification (1A, 2A, etc.) and their answers to the 40 written exam questions will be scanned into a file along with the key to the exam. Write a program that will read and score each contestant' s answers. It must then print a report that separates the contestants by school classification and ranks them based on their score on the test. The usual UIL scoring methods will be used. Contestants will earn 6 points for a correct answer, lose 2 points for an incorrect answer and `0` points for any question that is skipped. Ties will be broken using the percentage correct based on number of questions answered. The student with the greatest percent correct will win the tie.

Input:

The first line of the data file will be the word `KEY` followed by 40 letters `A` through `E` that represent the correct answers to the written exam each separated by a space. Next will be several lines of data each containing the following: first name, last name, school name (all will be one word), school classification and 40 capital letters `A` through `E` or an `S` if the question has been skipped. Each of the data items will be separated by a single space.

Output to the screen, a report that shows the results of the competition. The report should be grouped by school classification from 1A to 6A. Within each classification the contestants should be listed in descending order based on their test score each on a separate line. For each contestant show their placing, last name followed by a comma, their first name, their school's name, and their score on the exam. All items should be separated by a single space. Ties should be broken and displayed in the correct order with correct place. The tie breaker goes to the student with a higher percentage of correct answers. If two students have the same score and same percentage of correct answers, the tie is considered to be unbreakable. Unbreakable ties should be shown as having the same place and in alphabetical order based on last name, then first name. The next place after an unbreakable tie should reflect the same place, not the next place. For example, if there is an unbreakable tie for first place, list

both contestants as first and then list the next contestant as third place, not second. Let the user input the file name from the keyboard. For C++ use the `string` class. Refer to the sample output below.

Sample File:

```
KEY A E C B C B D C A E A C E B D E C D A C E C C B C D E E A C A B B B D B C D A D
Sebastian Williams Channing 1A A E C B C E D C E E E C E C E E C E A C E C D A B S S B A C A B E A D B C C E S
Alexander Taylor Gorman 1A A A S B A S D B S E C S E C S E D S A D E C B S C A E B B C A B S B B S C B S D
Jose Rodriguez Shamrock 1A S E C B C B D S A E E B C C D E C C A A A S E B B D E C A S S A E D D E S B A E
Angel Harris Rusk 4A A E D D C E B C C E A C A A B E A B B C E E B C D A A C A A D A B B C C E D A A
Sofia King Odessa 6A A E C B C B C C B E A C E C D B S B D S C D S D C E A C E S D D S B E S D D S S
Aria Morris Frost 2A C C C C C C C C S S S S S S S S S S C C C C C C C C C C C C C C C C C C C C C
Jacob Miller Zapata 4A A A D E E B B D B B E A B B D E C A E E E C C B A A D C C B C C D E E D C E E B
Michael Martinez Frost 2A A E D B E B D C A E E B D C D E E C A C B C C S C C S E B S D S S S S B C S A S
Emily Adams Alvin 6A A E C B C B D B A E E C E C D E C C A C E C D B C D E C A C A E E B C S S S S S S
Isaac Robinson Alvin 6A C E C A D B B C E E E B E B D E C A A D E A D E C A E C A D A A E E D C C A A A
Dylan Lopez Tulia 3A C A A B E E A B B C E D D B D B C C E D C E B D C B D D A B E E E D A D C B B D
Camila Green Poolville 2A A E C A C B D C A D E C E C D E C C A C E C D B C D S B S S A E E B D B S E A D
Penelope Stewart Van 4A D E C B C B D C C B E C E C D E C D E C E C D E C D E D A C A S E B S B C E A D
David Wilson Rusk 4A A E C B C B D C A E E C E C E E C C A C B D B E S C C E B A S C B E S E C S E C
Scarlett Turner Junction 2A A E D B E B D C A E E B D C D E E C A C B C C S C C S E B S D S S S S B C S A S
Jayden Jackson Poolville 2A A E C B C B D C A E E C E C D E C C A C E E A D E A D E D D C C D D B C C D B S
Elijah Davis Lasara 2A C E C B C B E C A E E C E C E E C E A C S C D S C D E E A C A B S S C D C B E A
Abigail Baker Lamesa 4A C C E C C B E B A E E C E D E S B C S S S S S S C C C S S S S S S S S S S S S S S
Natalie Flores Marfa 1A A E C B C B D C A E E C E C D S S S S S S S S S S S A A A A A S S S S S S S S S
Luke White Poolville 2A A E C B C B D A A E E C E C D S S S S S S S S S S S S S S S S S E E E E S S S S S
James Martin Hooks 3A A E C B C D D C C E E C E A D S B S S S E S S S S D E S S S S C E E S S S S S D
```

Sample Run:

```
Enter file name: contestants.txt

1A
1 Williams, Sebastian - Channing: 110
2 Flores, Natalie - Marfa: 80
3 Taylor, Alexander - Gorman: 76
4 Rodriguez, Jose - Shamrock: 76
2A
1 Green, Camila - Poolville: 136
2 Davis, Elijah - Lasara: 120
3 Martinez, Michael - Frost: 90
3 Turner, Scarlett - Junction: 90
5 Jackson, Jayden - Poolville: 90
6 White, Luke - Poolville: 58
7 Morris, Aria - Frost: 12
3A
1 Martin, James - Hooks: 74
2 Lopez, Dylan - Tulia: -16
4A
1 Stewart, Penelope - Van: 140
2 Wilson, David - Rusk: 80
3 Harris, Angel - Rusk: 40
4 Baker, Abigail - Lamesa: 16
5 Miller, Jacob - Zapata: 8
6A
1 Adams, Emily - Alvin: 138
2 Robinson, Isaac - Alvin: 72
3 King, Sofia - Odessa: 56
```

Name the program: `SchoolRankReportXX.java` or `SchoolRankReportXX.cpp`, where `XX` are your initials.

Required Problem- Comprehensive.

5(**). In any sport, scheduling the officials to work games in season can be a huge task. There are many factors involved, some that can be controlled others that cannot. In this problem, your task will be to write a program that schedules the 17 National Football League (NFL) officiating crews to 16 regular season games based on certain criteria. The criteria:

- No crew can work the same team (home or away) more than two times in the regular season.
- There must be a minimum of separation of six-weeks (6 or greater) before the crew can see the same team (home or away) again.
    - Example: Week 1 then again Week 6 or later.
- If a crew sees the same team twice during the season:
    - One must be at home and the other away- OR
    - Both times away
- Each crew (except one) will have two off or bye weeks in a 17-week season. Byes can't happen in consecutive weeks. Thus, each crew is guaranteed 15 games.
- No crew can work the same two matchups (home or away) during the season.
    - Example: `DAL @ NYG` and `NYG @ DAL`- even if all the other criteria stated above is met

Remember there are 17 weeks- but only 16 games in the NFL season. Also, due to team bye weeks, not all 32 NFL teams play every single week. Keep this mind when building crew schedules.

Input will come from a text file named `football20.txt`. The first 17 lines contain the referee's last name (crew chief). The remaining lines are a table consisting of the teams and weeks denoting each individual team and who they play each week either home, away, or a bye week. Output to a file in an easily readable format, two (2) possible regular season schedules for each of the 17 crews in a table with the crew and the teams they are scheduled to work each week. Output `OFF` if the crew is off that week. Finally, output a list of NFL teams they will not work in the season- if any. Let the user enter the input file name from the keyboard. Use user-defined functions/methods in your program. Do not use `break` or `continue` statements. For C++ use the `string` class. A linked list must be used with this problem. Output should be user friendly.

Name the program: `CrewSchedulingXX.java` or `CrewSchedulingXX.cpp`, where `XX` are your initials.

Extra Credit: Choose one of the two problems below for extra credit. See 2436 Grading Guide Sheet for additional grading/submission information. Partial credit will be given. No additional extra credit for implementing both programs.

1. Given a sequence of unsorted numbers, determine how badly out of order they are. Write a program that, for any given sequence of unique natural numbers, will compute the 'distance' between that original ordering and the same set of numbers sorted in ascending order. The distance should be computed by calculating how far displaced each number is in the original ordering from its correct location in the sorted list and summing those results. For instance, given the list `9 2 5 6 3`, the target list would be `2 3 5 6 9`. In this case, the 9 is four positions out of place, the 2 is one position out of place, the 5 and 6 are in the correct locations, and the 3 is three positions out of place. Therefore, the distance is `4 + 1 + 0 + 0 + 3 = 8`. Keyboard input will consist of a single integer, `m`, indicating the number of elements in the sequence. Followed on the next line the `m` integers in the list. Lists will contain at most 20 elements, each of which will be less than 100. For each sequence in the input, display the distance between the given ordering and its sorted counterpart. Refer to the sample output below.

Sample Runs:

```
Enter the number of elements in the sequence: 5
Enter the 5 integers: 9 2 5 6 3

The distance is: 8

Enter the number of elements in the sequence: 3
Enter the 5 integers: 1 49 99

The distance is: 0
```

Name the program: `SortDistanceXX.java` or `SortDistanceXX.cpp`, where `XX` are your initials.

2. A typical search process uses a sorted list and searches through the list for an item, or "key". The key is either in the list, or it is not. Write a program using an unsorted array of unique integers, within the range of -200 to 200, that sorts the list, then search for a given key to either find the position of the key or report the position where it should be in the list. For example, the list `22 33 25 81 -8 17 24 -27 95 100 -2` is sorted to become `-27 -8 -2 17 22 24 25 33 81 95 100`. To search for the key value 33, look through the list and find it, reporting its zero-based position, which is 7 in this list. If a key is not found in the list, like the value 2, report where it should have been found, position 3 in this list.
Input from the keyboard an array of integers (size 15 or less) all on one line, followed by an integer `n` on the next line indicating `n` search keys to follow, one on each line below that. Output the sorted array followed by a sentence for each search key reporting the result of the search, in the exact wording and format shown below. Refer to the sample output below.

Sample Run:

```
Enter array values: 22 33 25 81 -8 17 24 -27 95 100 -2

How many keys to search for: 5

Enter key 1: 33
Enter key 2: 2
Enter key 3: 100
Enter key 4: 86
Enter key 5: -27

Sorted array: -27 -8 -2 17 22 24 25 33 81 95 100

Key 1: 33 was found in position 7
Key 2: 2 was not found and should be in position 3
Key 3: 100 was found in position 10
Key 4: 86 was not found and should be in position 9
Key 5: -27 was found in position 0
```

Name the program: `SearchKeyXX.java` or `SearchKeyXX.cpp`, where `XX` are your initials.