Program Set #2
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- Stacks. Choose one problem.

1. It is your first day on the job at Facebizzle, and you cannot wait to sit down and write some code. However, Facebizzle uses the esoteric Flowjure language, which is a variant of LISP, and is heavy on the use of parentheses, brackets, and braces. Since you are new to writing Flowjure, you want to write a helper script to check if your Flowjure is valid and compute some statistics on our code. The most important characters in Flowjure are `(, [, {, }, ]`, and `)`. To be a valid Flowjure program, each the different types of parentheses must be balanced. Every opening parenthesis must be closed with the appropriate character in reverse order. For example, `([])` is balanced, but `{[}]` is not balanced, since the `[` is closed with the `}` character.
Input:
The file begins with an integer `T (1 ≤ T ≤ 10)`. After that, `T` test cases follow. Each test case begins with an integer `N, (1 ≤ N ≤ 30)`. `N` is the number of lines of code. After that follows `N` lines of Flowjure code. Flowjure code can contain any characters, but the script that you are writing ignores non-parenthesis characters.
Output:
Output to the screen each case label. If the code block is valid Flowjure, print "`YES- A () B [] C {}`", where `A`, `B`, and `C` are the number of parenthesis, square brackets, and curly braces respectively. If the code block is not valid Flowjure, print "`NO- X`" where `X` is the 0-based index of the first character that makes the code invalid. Ignore newlines when calculating this value. If it is impossible to detect that the code is invalid until the end of the program, print "`NO -1`". Let the user input the file name from the keyboard. If coding in C++, use C++ class string. Use a stack data structure. Refer to the sample output below.

Sample File
```
3
1
(map #(str "Hello " % "!" ) ["Ford" "Arthur" "Tricia"])
3
(apply map vector [[:a :b :c]
[:d :e :f]
[:g :h :i]])
2
def fib (cons 1 (cons 1
(lazy-seq (map + fib (rest fib))))))
```

Sample Run:

```
Enter file name: flojure.txt

Code Block 1: YES- 4 () 2 [] 0 {}
Code Block 2: YES- 2 () 8 [] 0 {}
Code Block 3: NO- 63
```

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

2. Palindromes are words or sentences containing 2 or more letters that read the same forward and backward. All the following are palindromes (note that punctuation, spacing, and upper/lowercase letter differences are ignored; only the letters themselves are considered).

```
       Racecar
       aa
       Madam, I'm Adam.
       A man, a plan, a canal. Panama.
```

The sentence below is not a palindrome, but it contains letters that form 3 different palindromes:

```
       Put the yellow piano on the truck.
```

Note that this sentence contains the letters `NOON` which form a palindrome, as well as the pair of letters `LL`, and the pair of letters `TT`. Write a program to find the longest palindrome contained in each sentence, marking its location on a separate line using square brackets to indicate the start and end of that palindrome. So, for example, given the piano sentence above, you would output the following:

```
       Put the yellow piano on the truck.
                         [   ]
```

A sentence may contain multiple palindromes that are tied for longest, in which case you would indicate the location of each of these palindromes on separate lines, in order of their occurrence, from left to right. For example:

```
       Madam made a yam for dinner, but that yam may be rotten.
         [     ]
                                             [     ]
```

For each input, find the longest palindrome(s). Any character that is not a letter should be ignored when determining whether text is a palindrome. Similarly, differences in upper/lowercase should be ignored (`A` and `a` are equivalent, for example). Note that the length of a palindrome is based only on the letters that comprise the palindrome. So, for example, `Radar`, `RA-DAR`, `r-a-d--a-r`, and `r475ad89a4r` all contain a palindrome of length 5. If a sentence does not contain a palindrome, then it should be output in its original form, with no additional lines of output. Note also that palindromes may overlap. For example, in the text "`abcdcbaxxxab`" there are two palindromes of length 7: "`abcdcba`" and "`baxxxab`". Since both palindromes are tied for "longest", both would be highlighted, beginning with the leftmost palindrome.

Input will be from the keyboard. Each input will be a string of at least 1 and at most 80 characters. The characters may include letters, numbers, spaces, or punctuation. Each of the input line will not contain leading or trailing spaces. Output the original sentence, then for each palindrome in that sentence that is tied for the longest palindrome, output a line indicating the beginning and end of that palindrome using square brackets, with an opening square bracket, `[`, below the letter where the palindrome begins, and a closing square bracket, `]`, below the letter where the palindrome ends. If there are multiple palindromes that are tied for longest, output a separate line for each, beginning with the leftmost longest palindrome, and proceeding from left to right. Do not output any spaces after the closing bracket. If a sentence contains no palindromes, then output just the sentence, but do not output any additional lines for that sentence. Use a stack data structure. For C++ use the `string` class. Refer to the sample output below.

Sample Runs:

```
Enter a palindrome: Madam, I'm Adam.

Madam, I'm Adam.
[             ]

Enter a palindrome: abcdcbaxxxab

abcdcbaxxxab
[     ]
     [     ]
```

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

Section Two- Queue/PQ. Choose one problem.

3. Your uncle has decided to expand the stack and queue process by adding some new processing features. Besides the normal push and pop, he adds low, high, and middle processes, either inserting a value or removing a value using these concepts. For working with stacks of digits, he decides on eight operations, which are as follows:

- `P` - Pop a digit from the top of the stack.
- `P(X)` - Push `X` onto the top of the stack.
- `L` - Remove the lowest valued digit from the stack.
- `L(X)` - Insert `X` immediately below the lowest valued digit in the stack.
- `H` - Remove the greatest valued digit from the stack.
- `H(X)` - Insert `X` immediately above the greatest valued digit in the stack.
- `M` - Remove the digit at the middle of the stack.
- `M(X)` - Insert `X` so that it is positioned as the new middle of the stack.

For queues of digits, he will use the same eight operations, as defined below:

- `P` - Pop a digit from the front of the queue.
- `P(X)` - Push `X` onto the back of the queue.

- `L` - Remove the lowest valued digit from the queue.
- `L(X)` - Insert `X` immediately in front of the lowest valued digit in the queue.
- `H` - Remove the greatest valued digit from the queue.
- `H(X)` - Insert `X` immediately behind the greatest valued digit in the queue.
- `M` - Remove the digit at the middle of the queue.
- `M(X)` - Insert `X` so that it is positioned as the new middle of the queue.

For a stack or queue with an even number of digits, your uncle observes that the "middle" could be one of two different digits. He decides that, for his purposes, "middle" shall refer to the digit from the middlemost pair that is closer to the bottom of the stack or the front of the queue. For example, in the list of digits below, the "middle" digit is the 6 (not the 1), whether the digits are in a stack or a queue:

```
Bottom/Front --> [5, 3, 7, 6, 1, 9, 8, 7] <-- Top/Back
```

Input:
An initial string of digits in the range 0-9, all on the first line, with single space separation. The values in this list are to be used to create both a stack and a queue by adding each digit in order, from left to right. On the next several lines will be commands that will either affect the stack or the queue, indicated by either the letter `S` or `Q`, followed by one or more commands to be evaluated left-to-right as defined above. When an `L` command is given, you may assume that the stack or queue contains only 1 digit that is strictly less than all other digits. Similarly, you may assume that when an `H` command is given, the stack or queue contains only 1 digit that is strictly greater than all others. No commands will result in attempting to remove a digit from an empty stack or queue.
Output:
After processing each line of commands, output to the screen the modified stack or queue, with the corresponding letter (`S` or `Q`) followed by its list of contents, as shown below. Stacks should be printed with the bottom of the stack to the left and the top of the stack to the right. Queues should be printed with the front of the queue to the left and the back of the queue to the right. Use a stack and a queue data structure. Refer to the sample output below.

Sample File

```
5 3 7 6 1 9 8 7
S P
Q P(4)
Q L M(2)
S L(0) H(2)
S L M
```

Sample Run:

```
Enter file name: process.txt

S [5, 3, 7, 6, 1, 9, 8]
Q [5, 3, 7, 6, 1, 9, 8, 7, 4]
Q [5, 3, 7, 6, 2, 9, 8, 7, 4]
S [5, 3, 7, 6, 0, 1, 9, 2, 8]
S [5, 3, 7, 1, 9, 2, 8]
```

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

4. A hospital emergency room operates as follows:

- When patients arrive, they are immediately evaluated and given a severity score, 1 through 10, with the higher numbers being more severe. Assume the evaluation doesn't take any time.
- Then the patient is asked to fill out paperwork that takes 5 minutes. Patients with severity 8 or higher can be seen before the paperwork is complete and will finish the paperwork after the doctor is done. Paperwork completed after seeing the doctor does not count toward the wait time.
- When a doctor is ready for a new patient, he grabs the highest severity person that has also been waiting the longest.
- The wait time starts when the patient arrives at the ER, includes the time spent completing the paperwork (as described in bullet #2 above) before seeing the doctor, and ends when the doctor starts seeing the patient.
- Assume that for any given severity, the doctor will spend the severity number times 8 minutes with the patient.
- The hospital will always have a minimum of 3 doctors on duty during the 24-hour period.

Write a program to create a simulation to help the hospital determine the optimal number of doctors that need to be on duty during a given 24 hour period so that the average wait time for all patients that enter the ER during that period is less than or equal to a given target wait time. You are to use the ER operations listed above to create the simulation. The hospital has given you several scenarios (described in the Input section below) to test your simulation. Input from a text file where first line will contain a single integer `n` that indicates the number of simulation scenarios to follow. For each scenario:

- The first line contains two integers `e` and `a`, separated by a space, where `e` is the number of entries in the simulation and `a (a ≥ 5)` is the target maximum wait time for a patient to be seen by a doctor, in minutes..
- The next `e` lines contain a time in 24-hour format, followed by a space and an integer `s (1 ≤ s ≤ 10)` denoting the severity of the patient.
- The entries in a scenario will be in the order patients arrive to the ER for a 24-hour period starting at `00:00` and ending at `23:59`.

For each scenario, output to the screen each scenario label and the minimum number of doctors that need to be on duty so the average wait time for the patients is less than or equal to the target wait time goal, followed by a space and the word `doctors`. Let the user input the file name from the keyboard. If coding in C++, use C++ class string. Use a queue/PQ data structure. Refer to the sample output below.

Sample File

```
1
8 5
00:01 9
00:01 9
00:01 3
00:06 8
11:01 3
11:15 8
11:30 8
23:10 1
```

Sample Run:

```
Enter file name: emergency.txt

Scenario 1: 4 doctors
```

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

Required Problem- Comprehensive.

5 (**). Write an interactive program for evaluating infix, prefix, and postfix expressions. The program will allow the user to enter an infix expression from the keyboard. Assume the expression is entered correctly. Output to the screen the prefix, infix, postfix, and final expression value. The infix output is placed in parenthesis to show the precedence of the operators explicitly. Finally, prompt the user if he/she wishes to perform another calculation or quit the program. Check case. The program must use a stack and/or queue data structure. Refer to the sample output below.

Sample Run:

```
Enter an infix expression: 4 + 5 * 2

Prefix: + 4 * 5 2
Infix: (4 + (5 * 2))
Postfix: 4 5 2 * +
Value: 14

Enter another expression (Y/N)? y

Enter an infix expression: (4 + 5) * 2

Prefix: * + 4 5 2
Infix: ((4 + 5) * 2)
Postfix: 4 5 + 2 *
Value: 18

Enter another expression (Y/N)? N
```

Name the program: `EvalExpressionsXX.java` or `EvalExpressionsXX.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. Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is used heavily in the IT world to pass data around from system to system. XML looks very much like HTML in that XML elements are encased in a set of less than and greater than signs. End tags are similar but have a slash at the beginning of the element name. Here is an example:

```
<MyElement>This is the text inside of a MyElement element!</MyElement>
```

XML tags can be named whatever you want, and they can be nested (which is what starts to unleash the true power of XML). Write a program to look through an XML file and list the element names that are found in order and the number of times that element name exists in the XML file. Remember do not count the ending tags, only the starting ones. Input from a data file will contain a snippet of XML. There will be no self-ending elements (i.e. `<ThisIsSelfEnding/>`). Elements may or may not have any content. The program should output to the screen the names of every XML element in the file in the order that they are found, along with the number of times that element exists in the file separated by a space. Let the user input the file name from the keyboard. If coding in C++, use C++ class string. Use a stack data structure. Refer to the sample output below.

Sample File:

```
<XML>
  <MoviesILike>
    <Movie>
      <Title>Back to the Future</Title>
      <Year>1982</Year>
    </Movie>
    <Movie>
      <Title>Iron Man</Title>
    </Movie>
  </MoviesILike>
  <MoviesIDoNotLike>
    <Movie>
      <Title>Steel Magnolias</Title>
    </Movie>
    <Movie>
      <Title>Grease</Title>
    </Movie>
  </MoviesIDoNotLike>
</XML>
```

Sample Run:

```
Enter file name: xmlelements.txt

XML 1
MoviesILike 1
Movie 4
Title 4
Year 1
MoviesIDoNotLike 1
```

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

2. There is a complete theory about how queues work. In this problem create a limited model to study the order in which a bunch of customers will be attended by their cashiers on a supermarket. The conditions for the experiment are:

- Each cashier spends the same amount of time with each customer (this is just an exercise, not real life).
- There will be a defined number of queues but never less than 2 and never more than 5.
- There is a variable number of customers, never less than 1 and never more than 20 and they are identified by a letter `(A, B, C, …)`
- The customers are distributed randomly among the different queues.
- If two customers are served at the same time, we would consider that they will be ordered following the queue number they are at (first will be customer in queue 1, second customer in queue 2)

Write a program to give the order in which the customers are attended, for example:

- There are 4 queues:
  - Cashier number 1 spends 3 minutes on each customer
  - Cashier number 2 spends 2 minutes on each customer
  - Cashier number 3 spends 4 minutes on each customer
  - Cashier number 4 spends 1 minutes on each customer
- Customers are distributed as follows:
  - Queue 1 (Cashier 1): `Customer A, customer E, customer I`
  - Queue 2 (Cashier 2): `B, F, J, N`
  - Queue 3 (Cashier 3): `C, G, L`
  - Queue 4 (Cashier 4): `D, H, M, O, P, Q`

With this input the customers will have been served in the following order and timing:

```
D (after 1 minute)
B (after 2 minutes on queue 2)
H (after 2 minutes on queue 4)
A (after 3 minutes on queue 1)
M (after 3 minutes on queue 4)
F (after 4 minutes on queue 2)
C (after 4 minutes on queue 3)
O (after 4 minutes on queue 4)
P (after 5 minutes)
E (after 6 minutes on queue 1)
J (after 6 minutes on queue 2)
Q (after 6 minutes on queue 4)
N (after 8 minutes on queue 2)
G (after 8 minutes on queue 3)
I (after 9 minutes)
L (after 12 minutes)
```

The input from a data file will have the number of queues on the first line, followed by the information for each queue, first the time spent by the cashier on a customer, the number of customers on a queue and then the order of the customers separated by a space. Output to the screen the list of served customers ordered by the time spent in the queue separated by spaces. For C++ use the string class. Must use a queue data structure. Refer to the sample output below.

Sample File:

```
4
3 3 A E I
2 4 B F J N
4 3 C G L
1 6 D H M O P Q
```

Sample Run:

```
Enter file name: queues.txt

The list ordered by time spent: D B H A M F C O P E J Q N G I L
```

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