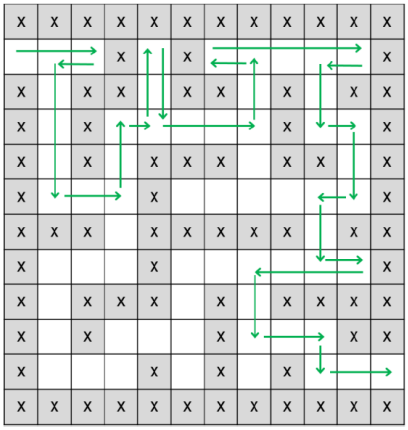
Program Set #2
Total Points: 30

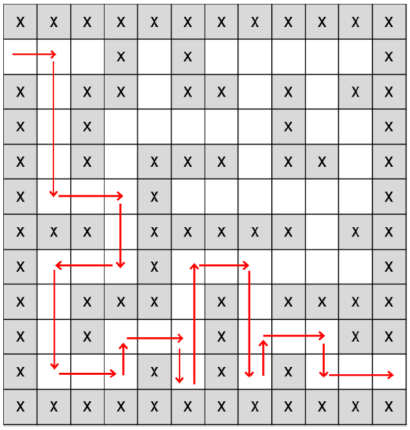
Three problems must be implemented for full credit. The required (starred) problem marked in the set must be implemented by everyone. See the 2436 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points each)

Note: If coding in C++, use the STL string class for problems involving strings. Do not use C style strings.

Section One- Choose two problems.

1. Write a program that finds the exit through a maze. There are two general rules to guarantee one can find the exit – the "left-hand" and "right-hand" rule. One can visualize this by placing their left or right hand on the nearest wall upon entering the maze. Then move through the maze without that hand ever losing contact with the wall until one reaches the exit. One will enter dead ends and then turn towards the right for the left-hand rule and toward the left for the right-hand rule. As one steps forward with an opening to the left for the left-hand rule, one would keep their hand on the corner of the wall and would turn to the left to continue. A similar move around a right corner for the right-hand rule allows one to stay in contact with the wall. The process continues until one arrives at the exit. One will never get lost and keep retracing previous paths. However, one can frequently return to a spot previously visited. In the example below, the mazes are the same, but the left maze uses the left-hand rule and right one uses right-hand rule.





For each maze, there will be exactly one entrance and one exit and there is always a solution. One can only move horizontally and vertically, no diagonal moves allowed, and the start and finish positions cannot be corners. The top-left corner is always position `(1,1)`.

Input will be from a data file where the first line of data file contains the number of test cases in the range `[1,20]`. For each test case, the first line contains three items separated by single spaces: the number of rows `R` and columns `C` in the maze, which will not exceed 30, followed by either `'L'` for the left-hand rule or `'R'` for the right-hand rule. The next line contains four integers separated by single

spaces: `r1 c1 r2 c2` where `(r1,c1)` is the start position and `(r2,c2)` is the finish position. The start and finish positions will not be adjacent to one another and will not be a corner of the maze. Both `r1` and `r2` will be in the range `[1,R]` and both `c1` and `c2` will be in the range `[1,C]`. The next `R` lines will each contain `C` characters separated by single spaces with no extra space at the end of the line. An `'X'` indicates that the position is blocked and cannot be entered while an `'O'` indicates that the position is empty and may be entered.

For each test case, output to the screen, the case label, the rule (`Left` or `Right`) then on the following lines output a list of positions visited in the form `"(r,c)"` where `r` and `c` are the row and column numbers with no spacing. Format the output from left to right with 10 positions per column right justified from start to end. Finally, output the total number of steps it took to reach the end, and display a blank line after each test case. Let the user input the file name from the keyboard. Finally, the program should ask if the user wants to run the program again (Check case). Use a stack data structure. Refer to the sample output below.

Sample File:

```
2
12 12 L
2 1 11 12
X X X X X X X X X X X X
O O O X O X O O O O O X
X O X X O X X O X O X X
X O X O O O O O X O O X
X O X O X X X O X X O X
X O O O X O O O O O O X
X X X O X X X X X O X X
X O O O X O O O O O O X
X O X X X O X O X X X X
X O X O O O X O O O X X
X O O O X O X O X O O O
X X X X X X X X X X X X
12 12 R
2 1 11 12
X X X X X X X X X X X X
O O O X O X O O O O O X
X O X X O X X O X O X X
X O X O O O O O X O O X
X O X O X X X O X X O X
X O O O X O O O O O O X
X X X O X X X X X O X X
X O O O X O O O O O O X
X O X X X O X O X X X X
X O X O O O X O O O X X
X O O O X O X O X O O O
X X X X X X X X X X X X
```

Sample Run:

```
Enter filename: maze.txt

Case 1: Left
 (2,1)  (2,2)  (2,3)   (2,2)   (3,2)   (4,2)   (5,2)  (6,2)  (6,3)  (6,4)
 (5,4)  (4,4)  (4,5)   (3,5)   (2,5)   (3,5)   (4,5)  (4,6)  (4,7)  (4,8)
 (3,8)  (2,8)  (2,7)   (2,8)   (2,9)  (2,10)  (2,11) (2,10) (3,10) (4,10)
(4,11) (5,11) (6,11)  (6,10)  (7,10)  (8,10)  (8,11) (8,10)  (8,9)  (8,8)
 (9,8) (10,8) (10,9) (10,10) (11,10) (11,11) (11,12)

47 Steps

Case 2: Right
 (2,1)   (2,2)   (3,2)   (4,2)   (5,2)  (6,2)  (6,3)  (6,4)  (7,4)  (8,4)
 (8,3)   (8,2)   (9,2)  (10,2)  (11,2) (11,3) (11,4) (10,4) (10,5) (10,6)
(11,6)  (10,6)   (9,6)   (8,6)   (8,7)  (8,8)  (9,8) (10,8) (11,8) (10,8)
(10,9) (10,10) (11,10) (11,11) (11,12)

35 Steps

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

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

2. Write a program that checks if an arithmetic expression with delimiters is nested correctly. The delimiters are parentheses ( ), brackets [ ], and braces { }. For an arithmetic expression to have correctly nested delimiters, the following two statements must be true:

- There are an equal number of right and left delimiters.
- Every right delimiter is preceded by a matching left delimiter.

For example, the expression `7 - ((x * ((x + y) / (j - 3)) + y) / (4 - 2.5))` is nested correctly because the two above conditions are satisfied. The expression `((a + b)` is not nested correctly because there are two left parentheses but only one right parenthesis, thus causing the first bullet above to not be met. The expression `{a * (b + c})` is not nested correctly either. Although there are equal numbers of right and left delimiters, the right brace } is preceded by a left parenthesis (, thus causing the second bullet above not to be met.

Input from the keyboard a mathematical expression. The expression may be composed of integer literals, variables, and the addition, subtraction, multiplication, and division operators, but the only delimiters allowed are parentheses, brackets, and braces. Assume no empty expressions. Output to the screen either:

`"The expression is nested correctly."` or

```
"The expression is not nested correctly."
```

on the following line. Finally, the program should ask if the user wants to run the program again (Check case). Use a stack data structure. Refer to the sample output below.

Sample Run:

```
Enter expression: {[(a + b) * 10 + c] * 10 + d}

The expression is nested correctly.

Run Again (Y/N): y

Enter expression: ((((((((((()))))))))))

The expression is not nested correctly.

Run Again (Y/N): y

Enter expression: a + b * d

The expression is nested correctly.

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

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

3. Palindromes are words or sentences containing two 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 the palindrome. For example, given the piano sentence above 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.
  [     ]
                                       [     ]
```

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), and 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 palindromes may also 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 between `[1,80]` characters. The characters may include letters, numbers, spaces, or punctuation. Assume each line will not contain leading or trailing spaces. Output to the screen 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. Finally, the program should ask if the user wants to run the program again (Check case). Use a stack data structure. Refer to the sample output below.

Sample Run:

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

Madam, I'm Adam.
[             ]

Run again (Y/N): Y

Enter palindrome: abcdcbaxxxab

abcdcbaxxxab
[     ]
```

```
     [      ]

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

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

Required Problem- Comprehensive.

4 (**). Write a program to balance chemical equations. Chemists obey the following rules when writing chemical equations:

- Each element's name is abbreviated by at most two letters. The first letter is always in upper-case and the second letter if exists, is a lower-case letter (Calcium is represented by `Ca`, Oxygen by `O`, and Chlorine by `Cl`).
- Each molecule is composed of a number of atoms. To represent a molecule, concatenate the abbreviated names of its composite atoms. For example, `NaCl` represents Sodium Chloride. Each atom name may be followed by a frequency number. For example, Calcium Chloride `CaCl2` consists of one atom of Calcium and two atoms of Chlorine. If the frequency is not given, it is assumed to be 1 (so `HCl` is equivalent to `H1Cl1`). There may be several occurrences of the same atom in the molecule formula, like `H` atom in `CH3COOH`.
- In ordinary chemical reactions, a number of molecules combine and result in a number of other molecules. For example, a well-known sample of neutralization is:

  `2HCl + CaO2H2 → CaCl2 + 2H2O`

  This means two molecules of chlorohydric acid (`HCl`) with one molecule of Calcium Hydroxide results in one molecule of Calcium Chloride (`CaCl2`) and two molecules of water.
- In every chemical reaction, the total number of each atom on the right side of the equation equals the total number of that atom on the left side.

Input is from a text file, where the first line contains an integer `N` between `[1,25]` the number of test cases. Each test case consists of a single line containing an expression like:

```
KMnO4 + HCl = KCl + MnCl2 + H2O + Cl2
```

OR

```
CaCl2 + AgNO3 = AgCl + Ca(NO3)2
```

For each labeled equation, output to the screen the balanced equation with the fewest number of integer coefficients in the format shown below. If an equation cannot be balanced correctly or uniquely, output `"Cannot Balance"`. Let the user input the file name from the keyboard. Finally, the program should ask if the user wants to run the program again (Check case). Use any appropriate data structure. Refer to the sample output below.

Sample File:

```
7
HCl + CaO2H2 = CaCl2 + H2O
HCl + H2SO4 = NaCl
KMnO4 + HCl = KCl + MnCl2 + H2O + Cl2
Cu + H2SOOO2 = CuSO2O2 + SO2 + H2O
H8 + S8 + FeFe2FeFe4 + Cl8Na8 = H4H3H2 + S9 + Fe9 + Cl9 + Na9
CaCl2 + AgNO3 = AgCl + Ca(NO3)2
CO2 = CO
```

Sample Run:

```
Enter filename: chemequations.txt

EQ1: 2HCl + CaO2H2 = CaCl2 + 2H2O
EQ2: Cannot Balance
EQ3: 2KMnO4 + 16HCl = 2KCl + 2MnCl2 + 8H2O + 5Cl2
EQ4: Cu + 2H2SOOO2 = CuSO2O2 + SO2 + 2H2O
EQ5: 9H8 + 9S8 + 9FeFe2FeFe4 + 9Cl8Na8 = 8H4H3H2 + 8S9 + 8Fe9 + 8Cl9 + 8Na9
EQ6: CaCl2 + 2AgNO3 → 2AgCl + Ca(NO3)2
EQ7: Cannot Balance

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

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

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

Write a program to capture all the pieces on a chess board and end up on the other side of the board. The rules:

- Every move must capture a new piece and all pieces must be captured.
- After capturing a piece, one can move and capture as the new piece, and can no longer move as the old piece.
- How chess pieces capture:
    - Pawns capture one space forward, in either diagonal.
    - Bishops capture diagonally, forward or backwards, in either direction, and can move any number of spaces.
    - Rooks capture straight ahead, or side to side (not diagonally), in either direction, and can move any number of spaces.
    - Knights capture in an L-shape: Some combination of 2 squares right/left and 1 square up/down, or 2 squares up/down and 1 square right/left in either order. Knights are the only piece that can "jump over" pieces while moving without capturing them.
    - Queens moves like combinations of bishops and rooks, so have 8 possible directions of movement for any number of spaces.
    - Kings move like queens but can only move one space.
- The player starts at the bottom edge of the board and must finish the sequence of moves at the top edge, with all pieces captured. Thus, the last move must be to capture the last piece on the top edge.
- A board configuration may not be solvable. If it is solvable, report a solution which is a legal sequence of piece captures that reaches the goal.

Input will be from a data file. The input will begin with a positive integer `N` in the range `[1,10]`, signifying the number of input instances. There will then follow `N` 8x8 grids of characters, separated by blank lines. Each element of the grid will be separated by a space. Each grid represents a chessboard. Each character in the grid will be either `'P'` (for Pawn), `'B'` (Bishop), `'R'` (Rook), `'N'` (Knight), `'Q'` (Queen), `'K'` (King), or `'-'` (empty space). There will be exactly one non-empty space on the bottom (the starting place and piece). For each input instance `i`, output to the screen either:

```
Board i: You should capture in this order: <path>
```

where `<path>` is the ordered sequence of pieces to capture; or

```
Board i: You are stuck!
```

Let the user input the file name from the keyboard. Finally, the program should ask if the user wants to run the program again (Check case). Use any appropriate data structure. Refer to the sample output

below.

Sample File

```
3
- - - - P - - -
- - - - - - - -
- - - - - N - -
- - - - - - - -
- R - - - - - B
- - - - - - - -
- - - - - - - -
- - - - B - - -

Q - - - - - - R
- - - - - - - -
- - - - - B - -
- - - N - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
R - - - - - - -

Q - - - - - - R
- - - - - - - -
- - - - - K - -
- - - N - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
R - - - - - - -
```

Sample Run:

```
Enter filename: cboard.txt

Board 1: You should capture in this order: BRBNP
Board 2: You should capture in this order: RQNBR
Board 3: You are stuck!

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

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