Question 1. (14 points) Character strings, linear search

Write a function `extractLastName` to extract a person's last name from a character vector having the following format: names appear in the order *first name, middle name(s), last name* and are separated from each other by one or more plus characters ('+'). Additionally, there may be extra plus characters after the last name.

The function `extractLastName` should accept a single input argument: a character vector containing the person's full name in the above format. It should return a single output: a character vector containing only the last name that appears in the input. Regardless of how many "names" appear in the input, whichever is last (farthest to the right) should be interpreted as the last name, even if it's the only name. If no name appears (i.e., the input is empty or all plus signs), then the returned character vector should be empty.

For example, if the input is '`Geralt+of+Rivia++`', then the returned value should be '`Rivia`'.

Solution 1: reverse linear search

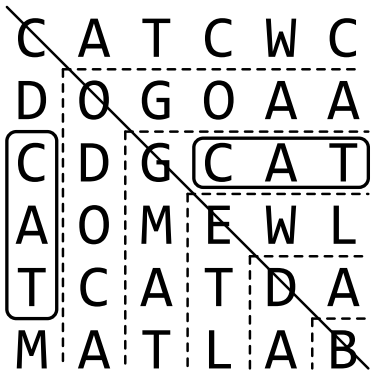
```
function lname = extractLastName(record)
% Return a character vector containing the last name in `record`, where
% name are delineated by plus signs.
k2 = length(record);  % Index of last non-plus character
while (k2 > 0) && (record(k2) == '+')
   k2 = k2 - 1;
end
k1 = k2;  % Index of last plus character before k2
while (k1 > 0) && (record(k1) ~= '+')
   k1 = k1 - 1;
end
lname = record(k1+1:k2);
```

Solution 2: reset between names

```
function lname = extractLastName(record)
lname = '';
for k = 1:length(record)
   if record(k) ~= '+'
      if k > 1 && record(k-1) == '+'
         lname = '';  % At start of word, so reset lname
      end
      lname = [lname record(k)];
   end
end
```

Question 2. (20 points) Triangular matrix traversal, transpose, string equality

Write a function `countWords` to search for a word in a square character matrix and count the number of times it occurs, subject to a few constraints: Words may be found horizontally (left-to-right) above the main diagonal of the matrix, or vertically (top-to-bottom) below the main diagonal of the matrix. Words may not use any letters on the main diagonal, nor may they be "backwards" (right-to-left or bottom-to-top).



The function `countWords` should take two input arguments: the first is the matrix to search in, and the second is the word to search for (a character row vector). It should return one output: the number of times that the word can be found.

Example: In this 6x6 character matrix, the word 'CAT' occurs twice according to the above rules, so the function should return 2. (It also occurs 3 other times in ways that violate the rules; these are not counted.)

```
function count = countWords(mat,word)
% Count the number of times `word` occurs in `mat`.
% Word may appear horizontally if right of main diagonal, or vertically
% if below main diagonal.
count = 0;
n = size(mat, 1);
w = length(word);
for i = 1:(n-w+1)  % No harm if this goes all the way to n
    for j = (i+1):(n-w+1)
        % Horizontal
        if strcmp(word, mat(i,j:(j+w-1)))
            count = count + 1;
        end
        % Vertical (transpose)
        % Note: Need to transpose vertical slice for strcmp to work
        if strcmp(word, mat(j:(j+w-1),i)')
            count = count + 1;
        end
    end
end
```

Question 3. (16 points) Image arrays, `uint8` arithmemtic, window iteration

Implement the following technique for making images look like old video. Every odd row of the image should be blurred horizontally, while every even row of the image should be set to black.

To blur the odd rows, each color intensity (red, green, and blue) of each pixel should be replaced by the average of itself and that of its left and right neighbors, 2 on each side (so nominally 5 pixels participate in the average). If a pixel doesn't have 2 neighbors on one side, use the average of itself and the neighbors it does have.

Name your function `makeLoFi`. It should accept one input argument, a 3-dimensional `uint8` array representing the original color image, and it should return one output value, a 3-dimensional `uint8` array of the same size representing the processed image.

```
function lofi = makeLoFi(hifi)
% Horizontally blur odd rows of color image `hifi` and set even rows to
% black.
[nr, nc, np] = size(hifi);
lofi = uint8(zeros(nr, nc, np));  % Even rows will be black by default
for r = 1:2:nr
   for c = 1:nc
       lb = max(1, c-2);  % Lower bound column of filter window
       ub = min(c+2, nc);  % Upper bound column of filter window
       w = hifi(r,lb:ub,:);  % Window slice
       % Sum will return type double, assignment will cast back to uint8
       % Sums over columns (because slice only contains 1 row)
       lofi(r,c,:) = sum(w)/length(w);
   end
end
```