Comments on Test 2A

Below are our comments from reading your work on Test 2A. Be sure to try to re-do any Test 2A question on which you didn't do well and approach any member of the course staff for help. Do not just read the solutions! (which will be posted later this week)

Regrade Requests

If you see a *grading error*--the grader mistook your correct code to be incorrect and deducted points--then please submit a regrade request on Gradescope and specify exactly what and where the grading error is. On the other hand, do not ask for a regrade if you dislike the number of points deducted--the same grading rubric was applied for all students and will not be changed.
You can submit a Test 2A regrade request on Gradescope *until 11am on Monday, 05/04*.

Question 1 (linear search)

Students did well:

- Most got the function header right and returned a char array properly
- Most people properly compared characters using `strcmp()` or `==`. Note that for individual characters, `==` is fine and preferred.
- A lot of people realized that traversing the input char array backwards would be easier and more efficient, so they either iterated through the input array from the back to the front or flipped the input array and started at the new 'beginning'

Could use improvement:

- A lot of student solutions did not generalize to edge cases and only worked for cases which looked like the example provided. Some solutions assumed the name had exactly three parts (first, middle, last), or did not account for variable numbers of + signs between or after names. Many people also assumed that the last name would always be preceded by a + sign, which is not necessarily the case if it only has one part.
- There were some issues initializing char arrays. Some students initialized an empty char array as `[]` rather than as `char()`, `''`, or `['']`. Many solutions also neglected to return an empty char array if the input array was empty.
- Most code was not particularly efficient and looped over the entire input array. If you loop backwards through the input array, you could stop the iteration when you've finished reading through the last name.
- Some students did not use `strcmp()` correctly, and tried to compare char arrays of two different lengths, which will always return false.
- Some students treated the input array as a 2D array rather than 1D. While this didn't always make the answer incorrect, it ended up making the code a lot more involved than necessary. Other students used cell arrays, which added some confusion and caused other mistakes in their code. Cell arrays are useful for storing lists of strings, but a single string is just a plain 1D char array.

Question 2 (matrix traversal)

Students did well:

- Most students got the function header right and many included helpful comments throughout their code - nice work!
- The accumulation pattern was implemented correctly in most cases
- A nested for-loop is indeed a good way to search over a matrix. Additionally, many students cleverly exploited the symmetry of the problem and wrote the function using only one set of for-loops

Could use improvement:

- Some students compared a row vector with a column vector when using `strcmp()` to search vertically. To resolve this issue, transpose one of the vectors.
- Be careful about when you perform a transpose and what indices you use with it. When a matrix is transposed, columns become rows, and the lower triangle becomes the upper triangle. If you want to iterate over the geometry of the original matrix, then consider only transposing after you've extracted a slice.
- Some students used == instead of `strcmp()` to compare two strings. While using == in an if-statement with equal-length vectors technically does the "right thing" here, it only works because MATLAB implicitly calls the `all()` function on the returned vector (which we have not discussed in class). Using `strcmp()` whenever comparing character vectors was and is strongly encouraged (since it works even when lengths are unequal), so we deducted a style point if == was used.
- When extracting a sub-array from the input matrix M, many students forget to constrain the indices to ensure that the slice does not go out of bounds.
- When setting the for-loop variables' lower and upper limits, many students are +1 or -1 off the correct value. It's best to use a small example to check the boundary values.
- A few students devised their own method of comparing strings, which unnecessarily complicates the problem (especially if it's not in a subfunction).

Question 3 (image processing)

Students did well:

- Most students got the correct function headers and assigned an output with the correct uint8 type.
- The nested for-loops were set up nicely with correct sizes in most cases.
- Many students were aware of the boundary cases and tried to set appropriate if-blocks to detect them.
- Most submissions were able to find or skip the even rows to turn them black correctly.

Could use improvement:

- Quite a lot of students did not consider the other boundary when handling a boundary case. For example, while they recognized that `c-2` would be a bad index when c is 1, they missed the fact that `c+2` could also be a bad index if the image is very thin (only 2 columns wide, for example). This would result in index-out-of-bounds errors for corner cases.
- Some students expected an image filename rather than a 3D uint8 array as input and therefore called `imread()` on the parameter. This violates the function specification and would crash if someone tried

to use it. Relatedly, there were also students who called `imshow()` at the end to draw the filtered image, which is a side effect not called for in the function requirements.

- There were a number of submissions that would lead to overflow when computing the average pixel due to how uint8 arithmetic works. There are several ways to do this right (sum() function, accumulate as doubles, divide each term, ...), but also several ways to do it wrong.
- Lastly, there were some submissions that replaced entries in the input array with averaged pixels rather than storing them in a separate output, which leads to incorrect blurring afterwards.

Statistics

Max: 100% (50/50)
Median: 88%
Mean: 82%
S.Dev: 17%

```
% Action to take after prelim
if  yourScore > 42/50

    toDo= celebrate + later look at the solution

elseif  yourScore > 32/50

    toDo= redo problems + later check solution

else
    toDo= meet with course staff to go over exam ...
          + stay caught up from now on ...
          + do NOT just read the solution--must redo problems
end
```

How does this impact my course grade?

See the Syllabus for grading details. This test is worth 8% of your grade, which is just slightly more than one of the later projects. The cutoff for a C- (S) is a total course grade of > 65.
After P5 and Test 2B we will have more detailed recommendations on how to estimate your standing in the course.