CS1112 Exercise 14 Name: _____ NetID: _____

This is an optional exercise BUT your grade for this exercise can replace your lowest exercise score. For example, if you missed or scored very low on one of the previous exercises, your score for this exercise can completely replace your score for that previous exercise (assuming you do better on this exercise). There is no MATLAB grader but you should *submit this exercise on CMS by Dec 6th.*

Review `removeChar`: pattern for divide-and-conquer problems

We developed `removeChar` and traced its execution during lecture. It was a divide-and-conquer problem on a vector. Here is a "recipe" that you can use for solving such divide-and-conquer problems. There are three steps:

1. **Deal with the "small data case(s)."** Solve the smallest sized problem. For problems on vectors, the small data cases are usually the length 0 and length 1 cases.
2. **Split the data into two parts and recurse.** The key is that each part has to be smaller than the original problem but together the parts cover the whole problem. On a vector problem, let's call one part left and the other right. You can split the data in different ways but a (usually) simple choice is to split off the first element as left and take all the remaining elements as right. Then recurse.
3. **Combine the results.** In the previous step, recursing on each part gives a result. Now you combine those two results into one answer.

Review the `removeChar.m` file from the exercise page to see several varitions of removing a character from a char array, all using the same 3-step recipt. This recipe is useful for helping beginners learn recursion, therefore we recommend that you use it. But recursion doesn't require this recipe, and so it is up to you whether to use it or not.

1 Function `numberOf`

Implement the following function as specified:

```
function count = numberOf(v, x)
% Returns the number of times scalar x occurs in vector v
% v: a numeric vector, possibly empty
% x: a numeric scalar
% count: an integer, the number of times that x occurs in v
% Use recursion.  No loops.
```

This is classic divide-and-conquer on a vector. The 3-step recipe will work well. Be sure to test your function! Start with these test cases:

- (Small data case) Empty vector: v=[] , x=5. The expected result is 0.
- (Small data case) Length 1 vector: v=2, x=5. The expected result is 0.
- (General case) Multiple occurrences: v=[4 3 4], x=4. The expected result is 2.

These are just initial tests. You should test your code more with other test cases before moving on to the next problems.

2 Function `removeDups`

Implement the following function as specified:

```
function w = removeDups(v)
% Returns vector v with adjacent duplicates removed
% v: a numeric vector, possibly empty
% w: v but with adjacent duplicates removed
% Example:  If v is [2,3,3,3,5,5,4,2,3,3]
%         then w is [2,3,5,4,2,3]
% Use recursion. No loops.
```

This is another classic divide-and-conquer on a vector. The 3-step recipe will work well, but you need to be careful with the last step that combines the results from the two parts left and right. Should you just concatenate the two results together like you did with `removeChar`? Consider what should happen when the last element in left and the first element in right are the same; what should happen when they are different?
Call your function a few times with some essential test cases first:

- (Small data case) Empty vector: removeDups([]) Expected result is []
- (Small data case) Length 1 vector: removeDups([2]) Expected result is 2 (same as [2])
- (Simple general case): removeDups([2,3,3,3,5]) Expected result is [2,3,5]

Again, use additional tests to fully verify that your code is correct.

3 Function `selectionSort`

Implement a sorting algorithm called selection sort, which will sort the contents of a numeric vector. The steps of the selection sort algorithm are as follows:

1. Find the smallest number, swap it with the first number in the array
2. Find the second smallest number, swap it with the second number in the array
3. Find the third smallest number, swap it with the third number in the array.
4. Repeat finding the next-smallest number, and swapping it into the correct position until the array is sorted.

This algorithm is called selection sort because it repeatedly selects the next-smallest element and swaps it into place. Use the function header provided below:

```
function v = selectionSort(v)
% sorts numeric vector v using the selection
% sort method
```

Submit `numberOf`, `removeDups`, and `selectionSort` on CMS.