CS112 Computation for the Sciences

Practice Exam 2

This exam is open book and open notes. There are 5 problems on the exam worth a total of 100 points. The number of points for each problem is shown below. **Try to do something on every problem, and show all of your work.**

Good luck!

Name _________________________________

Problem 1 _____________ (40/100 points)

Problem 2 _____________ (15/100 points)

Problem 3 _____________ (15/100 points)

Problem 4 _____________ (15/100 points)

Problem 5 _____________ (15/100 points)

TOTAL _______________
Problem 1: Looping with \texttt{for} and \texttt{while} (40 points)

An element of a vector can be removed by assigning the contents of its location to the empty vector, \([\ ]\). In the following example, the \texttt{nums} vector initially has five elements, but after executing the statement \texttt{nums(3) = [\ ]}, the third element is removed and the \texttt{nums} vector only contains four elements:

\begin{verbatim}
>> nums = [7 2 8 1 3]
noms =
   7   2   8   1   3
>> nums(3) = []
noms =
   7   2   1   3
\end{verbatim}

The \texttt{removeDuplicates} function, as defined below, contains bugs. The intent of this function is to remove duplicates of any values that are repeated in the input \texttt{nums} vector, and return the modified vector with the duplicates removed:

\begin{verbatim}
function removeDuplicates (nums)
   n = length(nums);
   for i = 1:n
      if (any(nums(1:i-1) == nums(i)))
         nums(i) = [];
      end
   end
\end{verbatim}

The following is an example of the intended behavior of the \texttt{removeDuplicates} function:

\begin{verbatim}
>> nums = [2 1 7 8 2 6 7 7 3 8];
>> newNums = removeDuplicates (nums)
   2   1   7   8   6   3
\end{verbatim}

Part a (10 points): Describe the errors in the current definition of \texttt{removeDuplicates} and modify the definition to fix these errors. Your revised function should still contain a \texttt{for} loop.
Part b (15 points): Rewrite the `removeDuplicates` function using a `while` statement instead of a `for` statement, to loop through the input `nums` vector.
Part c (15 points): Hand simulate the mystery function defined below, and indicate what is printed by the following sequence of statements:

```matlab
>> nums = [8 2 7 3 6];
>> newNumbers = mystery(nums)
>> nums
```

```matlab
function newnums = mystery(nums)
for i = length(nums):-1:2
    [val index] = max(nums(1:i));
    temp = nums(i);
    nums(i) = nums(index);
    nums(index) = temp;
end
newnums = nums;
```

What does the mystery function do, in general?
Problem 2: Nested for loops (15 points)

Write a function `sumMX` that has a single input that is a 2-D matrix that is assumed to contain all 0’s and 1’s, and a single output that is a 2-D matrix of the same size. This function should count the number of 1’s contained in a 3x3 block centered at each location of the input matrix, and store this sum at the corresponding location of the output matrix. You do not need to calculate these sums for the locations around the border of the image (the first and last row and column) – the output matrix can contain zeros at these locations. Given the following input matrix:

```
1 0 1 1 0
0 1 0 1 1
1 0 0 1 0
0 1 0 0 1
1 1 1 0 1
```

The `sumMX` function should return the following output matrix:

```
0 0 0 0 0
0 4 5 5 0
0 3 4 4 0
0 5 4 4 0
0 0 0 0 0
```
Problem 3: Function inputs and outputs (15 points)

A student has written the following functions (using very poor choices of variables and no comments) in an M-file named `myFunction.m`.

```matlab
function [a b c] = myFunction(a, b, c)
    [c b a] = mySubFunction(a, b);

function [a b c] = mySubFunction(x, y, z)
    if (nargin == 2)
        z = 4;
    end
    nums = [x y z];
    a = max(nums);
    c = min(nums);
    b = nums((nums ~= a) & (nums ~= c));
```

What are the values returned in the variables `a`, `b`, and `c` when the following command is typed into MATLAB’s Command Window?

```matlab
>> [a b c] = myFunction(2, 1, 3);
```
Problem 4: Reading files from folders (15 points)

Suppose the Current Directory contains a folder named images that stores many image files with a .jpg file extension. Write a function named countLargeImages that reads in each image file in the images folder and counts the number of images that have at least one dimension whose size is greater than 100. The countLargeImages function should have no inputs and should return the total number of large images.
Problem 5: Structures (15 points)

The following picture portrays the contents of a structure named claire that stores assignment, exam and project grades for a student in a course:

```
<table>
<thead>
<tr>
<th>claire</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignments: 82 97 90 85 89</td>
</tr>
<tr>
<td>exams: 83 91</td>
</tr>
<tr>
<td>projects: 95 93</td>
</tr>
</tbody>
</table>
```

**Part a (9 points):** Define a function `addGrade` that has three inputs: (1) a structure that contains the fields shown above, (2) the name of a category specified as a string, e.g. ‘assignment’, ‘exam’ or ‘project’, and (3) a new grade. The `addGrade` function should return the structure with the new grade added to the end of the vector of grades in the specified category.

**Part b (6 points):** Write a script named `testGrade` that contains code to create the structure portrayed in the above picture and then calls the `addGrade` function three times, to add a new assignment grade of 94, a new exam grade of 87 and a new project grade of 91.
Problem 1: Looping with \texttt{for} and \texttt{while}

The following buggy code was given:

\begin{verbatim}
function removeDuplicates (nums)
    n = length(nums);
    for i = 1:n
        if (any(nums(1:i-1) == nums(i)))
            nums(i) = [];
        end
    end
end
\end{verbatim}

\textbf{Part a:} There are two errors in this function. First, the function should return the modified \texttt{nums} vector, but there is no output parameter specified in the header of the function. Second, the control variable for the loop, \texttt{i}, spans the values from 1 to \texttt{n}, where \texttt{n} is the length of the original \texttt{nums} vector. As elements are removed from \texttt{nums}, it becomes shorter, so eventually an out-of-bounds error is encountered, when the index \texttt{i} exceeds the new length of the \texttt{nums} vector. When \texttt{i} is 1, the expression \texttt{1:i-1} evaluates to \texttt{1:0}, which is the empty vector, but this does not actually generate an error. The following definition fixes the errors in this function, preserving the use of a \texttt{for} statement:

\begin{verbatim}
function newNums = removeDuplicates (nums)
    n = length(nums);
    newNums = [];
    for i = 1:n
        if ~(any(nums(1:i-1) == nums(i)))
            newNums = [newNums nums(i)];
            % alternative: newNums(end+1) = nums(i);
        end
    end
end
\end{verbatim}

\textbf{Part b:} Here is one way to implement the function using a \texttt{while} loop:

\begin{verbatim}
function newNums = removeDuplicates (nums)
    n = length(nums);
    newNums = [];
    i = 1;
    while (i <= n)
        if ~(any(nums(1:i-1) == nums(i)))
            newNums = [newNums nums(i)];
        end
        i = i + 1;
    end
end
\end{verbatim}
Part c: The function mystery sorts the numbers contained in the input nums vector in increasing order, using a sorting strategy known as selection sort. Suppose there are \( n \) numbers in the input vector. The selection sort algorithm first places the maximum value into the last location (index \( n \)) of the vector. Then the second largest value is placed at index \( n-1 \), the third largest value is placed at index \( n-2 \), and so on. When a number is moved from an index \( i \) to its final place at the index given by index, the contents at index are moved down to index \( i \). When mystery is called with the vector \([8 \ 2 \ 7 \ 3 \ 6]\), the contents of the nums vector after each iteration of the for loop are as follows:

initial values: \([8 \ 2 \ 7 \ 3 \ 6]\)
i = 5: \([6 \ 2 \ 7 \ 3 \ 8]\)
i = 4: \([6 \ 2 \ 3 \ 7 \ 8]\)
i = 3: \([3 \ 2 \ 6 \ 7 \ 8]\)
i = 2: \([2 \ 3 \ 6 \ 7 \ 8]\)

This results in the following printout from the specified commands:

```matlab
>> nums = [8 2 7 3 6];
>> newNumbers = mystery(nums)
newNumbers =
  2   3   6   7   8
>> nums
nums =
  8   2   7   3   6
>>
```

Problem 2: Nested for loops

The following is one possible definition of sumMX:

```matlab
function outMX = sumMX(inMX)
[rows cols] = size(inMX);
outMX = zeros(rows, cols);
for row = 2:rows-1
    for col = 2:cols-1
        outMX(row, col) = sum(sum(inMX(row-1:row+1, col-1:col+1)));
    end
end
```

Problem 3: Function inputs and outputs

The following printout shows the values returned in the variables \( a \), \( b \), and \( c \) when the following command is typed into MATLAB’s Command Window:
>> [a b c] = myFunction(2,1,3)
    a =
        1
    b =
        2
    c =
        4

Problem 4: Reading files from folders

Here is one implementation of the countlargeImages function:

```matlab
function count = countLargeImages
    folderName = [pwd filesep 'images' filesep];
    fileNames = dir(fullfile(folderName, '*.jpg'));
    count = 0;
    for i = 1:length(fileNames)
        im = imread(fullfile(folderName, fileNames(i).name));
        if (max(size(im)) >= 100)
            count = count+1;
        end
    end
end
```

Problem 5: Structures

The addGrade function can be defined as follows:

```matlab
function newStructure = addGrade(oldStructure, category, grade)
    if strcmp(category, 'assignment')
        oldStructure.assignments(end+1) = grade;
    elseif strcmp(category, 'exam')
        oldStructure.exams(end+1) = grade;
    else
        oldStructure.projects(end+1) = grade;
    end
    newStructure = oldStructure;
end
```

The following script creates the claire structure and tests the addGrade function:

```matlab
% testGrade.m script
claire.assignments = [82 97 90 85 89];
claire.exams = [83 91];
claire.projects = [95 93];
claire = addGrade(claire, 'assignment', 94);
claire = addGrade(claire, 'exam', 87);
claire = addGrade(claire, 'project', 91);
```