CS1112 Fall 2019 Prelim 2 Nov 14 7:30–9:00pm

Name: ____ NetID: ____________

(Legibly print last name, first name, middle name)

Statement of integrity: *It is a violation of the Code of Academic Integrity to look at any exam other than your own, to look at any other reference material, or to otherwise give or receive unauthorized help. Understanding this, I declare I shall not give, use, or receive unauthorized aid in this examination.*

____

(Signature)

Circle your lecture time: 9:05 or 11:15

Circle your discussion instructor's name:

	Tuesday	Wednesday
10:10		Noam Eshed
11:15		Noam Eshed
12:20	Zhilong Li/June Cho	Joseph Kim
1:25	Vaishnavi Dhulkhed	Joseph Kim
2:30	Darian Nwankwo	Darian Nwankwo
3:35	Darian Nwankwo	Zhilong Li

Instructions:

- This is a 90-minute, closed-book exam; no calculators are allowed.
- The exam is worth a total of 100 points, so it's about one point per minute!
- Read each problem completely, including any provided code, before starting it.
- Raise your hand if you have any questions.
- Clarity, conciseness, and good programming style count for credit.
- If you supply multiple answers, we will grade only one.
- Use only Matlab code. No credit for code written in other programming languages.
- Assume there will be no input errors.
- Do not modify given code unless instructed to do so.
- Do not write user-defined functions or subfunctions unless instructed to do so.
- Do not use `switch`, `try`, `catch`, `break`, `continue`, or `return` statements.
- Do not use built-in functions that have not been discussed in the course.
- You may find the following Matlab predefined functions useful:
  `abs`, `sqrt`, `rem`, `min`, `max`, `floor`, `ceil`, `rand`, `zeros`, `ones`, `sum`, `length`, `size`, `isempty`, `fprintf`, `disp`, `uint8`, `double`, `char`, `strcmp`, `str2double`, `fopen`, `fclose`, `fgetl`, `feof`, `cell`, `struct`

Examples:
`zeros(1,4)` → 1 row 4 columns of zeros, type `double`
`cell(3,2)` → a 3-by-2 cell array, each cell is the empty numeric vector `[]`
`length([2 4 8])` → 3, length of a vector
`[nr,nc,np]=size(M)` → dimensions of `M`: `nr` rows, `nc` columns, `np` layers
`strcmp('cat','Cat')` → 0, the two strings are not identical
`str2double(' -2.6 ')` → -2.6, a type `double` scalar
`uint8(4.7)` → the integer (type `uint8`) value 5
`struct('a',1,'b',0)` → a structure with 2 fields: `a` has value 1, `b` has value 0
Allowed by some questions:
`min(-4,3)` → -4, smallest argument
`max(-4,3)` → 3, largest argument
`sum([0 4; 1 -1])` → [1 3], vector of column sums of the *matrix* argument

Question 1 (8 points)

In the chart below, there are four columns for four kinds of arrays that we use often. For each row there is a statement on the left that is either true or false depending on the kind of array. In each box of the chart marked "T F", circle *either* **T** for true *or* **F** for false.

	Numeric array (double, uint8)	char array	Cell array	Struct array
Initialized using {}	T [✓ F]	T [✓ F]	[✓ T] F	T [✓ F]
Indexed using [ ]	T [✓ F]	T [✓ F]	T [✓ F]	T [✓ F]
All elements must have the same type	[✓ T] F	[✓ T] F	T [✓ F]	[✓ T] F
At each index store exactly one scalar value	[✓ T] F	[✓ T] F	T [✓ F]	▨
Can be 1-d, 2-d, or 3-d	[✓ T] F	[✓ T] F	[✓ T] F	[✓ T] F

Question 2 (12 points)

(2.1) Write one expression on each of the two blanks below such that the script prints appropriate output.

```
dna= ['acgtt'; ...
      'tacca'; ...
      'aaacg'; ...
      'ggggt'];

for r= 1:4
    c= 1;
    while  c <= 5  &&  dna(r,c)~='g'
        c= c+1;
    end

    if __ c<=5 __ % Alternative:  c<6
                  % Alternative:  c<6 && dna(r,c)=='g'
                  % Incorrect:    dna(r,c)=='g' && c<6  index out of bounds error
                  % Incorrect:    dna(r,c)=='g'         index out of bounds error

        fprintf('On row %d, found first ''g'' at column %d.\n', r, _____ c ______)
    else
        fprintf('On row %d, ''g'' not found.\n', r)
    end
end
```

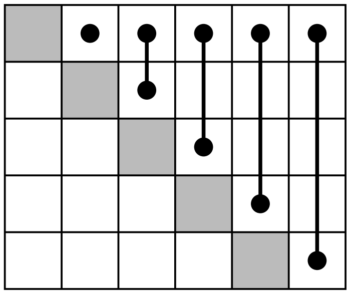
(2.2) When the script from part **(2.1)** above is run, how many times is the expression dna(r,c)~='g' in the loop guard of the while-loop evaluated? (I.e., how many times is the relational operation ~= in the loop guard executed?)

14 (=3+5+5+1)

Question 3 (10 points)

Fill in the placeholders `A` through `F` with variable names and expressions so that the code below performs a *column-major* traversal of the elements of matrix `M` above and to the right of, but excluding, the diagonal, as illustrated below (do not make any assumptions about the size of `M` based on the illustration). The built-in functions `min` and `max` are allowed in your expressions.

```
[nr, nc] = size(M);

for A = B:C

   for D = E:F

      disp(M(r,c))

   end

end
```



Solution

- A: `c`
- B: `2` (`1` is also acceptable)
- C: `nc`
- D: `r`
- E: `1`
- F: `min(c-1, nr)`

Question 4 (15 points)

In class you have worked with images in the *RGB* colorspace, where channel 1 is R (red), channel 2 is G (green), and channel 3 is B (blue). But images can be represented in other colorspaces. Consider the *YCoCg* colorspace, where the channels Y, C_o, and C_g are mathematically related to R, G, and B via the following equations:

$$\begin{aligned} Y &= \frac{1}{4}(2G + R + B) \\ C_o &= \frac{1}{2}(R - B) + 128 \\ C_g &= \frac{1}{4}(2G - R - B) + 128 \end{aligned}$$

(this assumes that R, G, and B are in the range 0-255 and will yield Y, C_o, and C_g in that same range).

Implement the following function (hint: remember the rules of `uint8` arithmetic):

```
function ycocg = rgb2ycocg(rgb)
% Transform image from RGB colorspace to YCoCg colorspace.
% rgb is a 3D uint8 array such that rgb(i,j,k) is the value of channel
%   k (R, G, or B) for the pixel at row i and column j.
% ycocg is a 3D uint8 array such that ycocg(i,j,k) is the value of
%  channel k (Y, Co, or Cg) for the pixel at row i and column j after
%  being transformed from RGB to YCoCg.
```

Solution

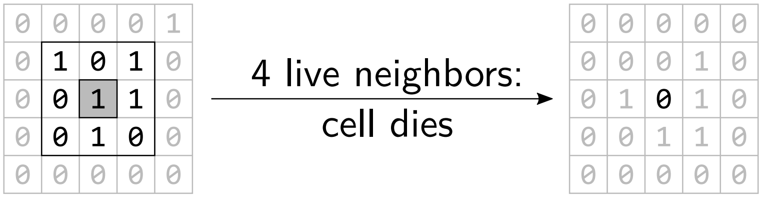
```
% Example vectorized solution
r = double(rgb(:,:,1));
g = double(rgb(:,:,2));
b = double(rgb(:,:,3));
ycocg(:,:,1) = uint8((2*g + r + b)/4);
ycocg(:,:,2) = uint8((r - b)/2 + 128);
ycocg(:,:,3) = uint8((2*g - r - b)/4 + 128);

% Example non-vectorized solution
for i = 1:size(rgb,1)
   for j = 1:size(rgb,2)
      ycocg(i,j,1) = uint8((2*double(rgb(i,j,2)) + ...
                           double(rgb(i,j,1)) + ...
                           double(rgb(i,j,3)))/4);

      ycocg(i,j,2) = uint8((double(rgb(i,j,1)) - ...
                           double(rgb(i,j,3)))/2 + 128);

      ycocg(i,j,3) = uint8((2*double(rgb(i,j,2)) - ...
                           double(rgb(i,j,1)) - ...
                           double(rgb(i,j,3)))/4 + 128);

      % Tricky no-double solution (order-of-operations matters)
      % ycocg(i,j,1) = rgb(i,j,2)/2 + rgb(i,j,1)/4 + rgb(i,j,3)/4;
      % ycocg(i,j,2) = 128 + rgb(i,j,1)/2 - rgb(i,j,3)/2;
      % ycocg(i,j,3) = 128 + rgb(i,j,2)/2 - rgb(i,j,1)/4 - rgb(i,j,3)/4;
   end
end
```

Question 5 (20 points)

Write a function to compute one step in a simulation known as “Conway’s Game of Life.” The game board is a 2D array where each element is either a `1` (living cell) or a `0` (dead cell). Elements have up to 8 neighbors in the array: above & below, left & right, and 4 diagonals (in other words, a block of radius 1, excluding the center). To simulate one generation, the board is updated according to these rules:

1. Any living cell with fewer than 2 living neighbors dies (underpopulation)
2. Any living cell with 2 or 3 living neighbors continues to live
3. Any living cell with more than 3 living neighbors dies (overpopulation)
4. Any dead cell with exactly 3 living neighbors becomes alive (reproduction)

Would-be neighbors outside of the array bounds do not count as being alive. Example:



0	0	0	0	1
0	1	0	1	0
0	0	1	1	0
0	0	1	0	0
0	0	0	0	0

`board`

0	0	0	0	0
0	0	0	1	0
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0

`newBoard`

Implement the following function to perform this update, storing the results in a new array. You may use the built-in functions `min`, `max`, and `sum` if you like.

```
function newBoard = nextGeneration(board)
% Given board, a 2D type double array of 1s and 0s (representing living and dead
% biological cells), simulate one generation of Conway's Game of Live and return
% the new state of the board in newBoard (also a 2D type double array of 1s and 0s).
```

Solution

```
[nr, nc] = size(board);
newBoard = zeros(nr, nc);
for r = 1:nr
   for c = 1:nc

      neighbors = 0;
      for rr = max(1,r-1):min(nr,r+1)
         for cc = max(1,c-1):min(nc,c+1)
            neighbors = neighbors + board(rr,cc);
         end
      end
      neighbors = neighbors - board(r,c);

      % There are many different, correct ways to express this logic;
      % here is one example
      if (board(r,c) == 1) && (neighbors < 2)
         newBoard(r,c) = 0;
      elseif (board(r,c) == 1) && (neighbors > 3)
         newBoard(r,c) = 0;
      elseif (board(r,c) == 0) && (neighbors == 3)
         newBoard(r,c) = 1;
      else
         newBoard(r,c) = board(r,c);
      end
   end
end
```

Question 6 (35 points)

We have a plain text file containing data on many cities in New York state. Each line of the file stores the data of one city, in the order name, latitude, longitude, and population. The data items on a line are separated by exactly two spaces. As an example, two of the many lines from the file are shown below with the symbol ␣ indicating a single space:

```
New␣York␣␣40°39′40″N␣␣73°56′38″W␣␣8175133
Ithaca␣␣42°26′36″N␣␣76°30′0″W␣␣30999
```

(6.1) Complete the following function as specified:

```
function D = parseData(cityData)
% Read text from the file named by cityData and return a 2D cell array.
% Each line of the file contains information on one city:  name, latitude,
%   longitude, and population.  Those 4 data items are separated by exactly 2
%   spaces. There are no leading or trailing spaces on each line.
% D is a n-by-4 cell array where n is the number of lines of text in the file,
%   and each cell in one row of D stores one data token for a city.  The city
%   name, latitude, and longitude are each stored in D as a char row vector; the
%   population is stored in D as a type double scalar.
% Example: suppose one line from the file is
%   'New␣York␣␣40°39′40″N␣␣73°56′38″W␣␣8175133'.  Then the row in cell array D
%   that corresponds to that line stores 'New York' in the 1st cell, '40°39′40″N'
%   in the 2nd cell, '73°56′38″W' in the 3rd cell, and 8175133 in the 4th cell.
%   There must not be any leading or trailing spaces in the char vectors.
```

Solution

```
fid= fopen(cityData, 'r');

r= 1;  % current row number

while ~feof(fid)

    s= fgetl(fid);  % s is a char row vector containing one line from the file

    c= 1;       % which token
    tStart= 1;  % start index of current token
    % Get first 3 tokens
    for k= 1:length(s)-1
        if strcmp(s(k:k+1), '  ')
            tEnd= k-1;
            D{r,c}= s(tStart:tEnd);
            c= c+1;
            tStart= k+2;
        end
    end
    % Last token
    D{r,4}= str2double( s(tStart:length(s)) );
    r= r+1;

end
fclose(fid);
```

[Question 6, continued]

(6.2) Assume the availability of the following function:

```
function city = makeCity(name, latitude, longitude, population)
% Return a city struct.  Parameters name, latitude, longitude are each a
% char row vector.  Parameter population is a type double scalar.
city= struct('name',name,'lat',latitude,'lon',longitude,'pop',population);
```

By making effective use of the given function makeCity, implement the following function as specified:

```
function sa = cells2structs(D)
% Store data of cell array D, the city data as returned by function parseData,
% in sa, an array of structs.  Each struct in sa is a city struct as given in
% function makeCity.  Make effective use of function makeCity.
```

Example solution

```
[nr,nc]= size(D);
for r= 1:nr
    sa(r)= makeCity(D{r,1}, D{r,2}, D{r,3}, D{r,4});
end
```

(6.3) Implement the following function as specified; do not use the built-in functions min, max, or sort:

```
function printSmallest(ctys)
% Print the name and population of the smallest (by population) cities.
% ctys is a 1-d struct array; each struct in ctys has the fields name, lat, lon,
% and pop, as given by the function makeCity.  If multiple cities are tied for
% the smallest population, print the name and population of all of them.
```

Example solutions

```
%% Example 1:  Store the indices of smallest cities; then print
smallestPop= ctys(1).pop;
iSmallest= 1;  % indices of cities with lowest pop
for k= 2:length(ctys)
    if ctys(k).pop < smallestPop
        smallestPop= ctys(k).pop;
        iSmallest= k;
    elseif ctys(k).pop == smallestPop
        iSmallest= [iSmallest k];
    end
end
for s= 1:length(iSmallest)
    fprintf('%s \t %15d\n', ctys(iSmallest(s)).name, ctys(iSmallest(s)).pop)
end

%% Example 2:  Find smallest pop; then loop again to print the cities with that pop
smallestPop= ctys(1).pop;
for k= 2:length(ctys)
    if ctys(k).pop < smallestPop
        smallestPop= ctys(k).pop;
    end
end
for k= 1:length(ctys)
    if ctys(k).pop == smallestPop
        fprintf('%s \t %15d\n', ctys(k).name, ctys(k).pop)
    end
end
```