Name: Example Solutions 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 reference material outside of this exam packet, to communicate with anyone other than the exam proctors, or to otherwise give or receive any unauthorized help during the exam.

Academic Integrity is expected of all students of Cornell University at all times. **By submitting this exam, you declare that you will not give, use, or receive unauthorized aid in this examination.**

Circle your discussion section:

	Wednesday	Thursday
9:40 AM	Aravind Suresh Babu	Aravind Suresh Babu
11:25 AM	Subham Sahoo	Subham Sahoo
1:00 PM	Claire Liang	-
2:45 PM	Claire Liang	-

Instructions:

- Check that this packet has 7 double-sided sheets.
- 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.
- Do not modify any *given* code unless asked to do so.
- Raise your hand if you have any questions.
- Use the back of the pages if you need additional space.
- Clarity, conciseness, and good programming style count for credit.
- Indicate your final answer. If you supply multiple answers, you may receive a *zero* on that question.
- Use only MATLAB code. No credit for code written in other programming languages.
- Assume there will be no input errors.
- Write user-defined functions and subfunctions only if asked 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`, `fprintf`, `disp`, `uint8`, `double`, `char`, `strcmp`, `str2double`, `cell`

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

Question 1 (17 points)

(1.1) Write the output given by each disp statement below.

```
x = {'a' 'dog'};
disp( length(x) )              % What is the output?   ANSWER: 2


disp( length([x{1} x{2}]) )    % What is the output?   ANSWER: 4


disp( length({x{1} x{2}}) )    % What is the output?   ANSWER: 2
```

(1.2) Assume that variables r and s each stores a type uint8 value. Complete the blank to assign to variable d the difference (absolute value) between r and s. d should have the type uint8.

Example solutions:

```
d = (r-s) + (s-r)
d = uint8( abs(double(r) - double(s)) )
d = max(r,s) - min(r,s)
d = max(r-s, s-r)
```

(1.3) Complete the following function as specified:

```
function h = distHistogram(z)
% z is a type double matrix storing coordinates of some points on the Cartesian
%   plane.  z has two columns:  column 1 stores x-coordinates; column 2 stores
%   y-coordinates.  Each row represents one point.  z has at least one row.
% h is the data (vector) for drawing a bar graph showing the distribution of
%   distance from the origin:  h(1) is the number of points a distance 1 or less
%   from the origin, h(2) is the number of points whose distance is >1 and <=2 from
%   the origin,..., h(100) is the number of points whose distance is >99 and <=100
%   from the origin. Do not count the points whose distance from the origin is >100.

h= zeros(1,100);
x= z(:,1);
y= z(:,2);
dist= sqrt(x.^2 + y.^2);
% Add your code below
```

Example solutions:

```
% Example solution
n= size(z,1);
for k = 1:n
    binNum= ceil(dist(k));
    if  binNum<=100
        if binNum==0
            binNum= 1;
        end
        h(binNum) = h(binNum) + 1;
    end
end



bar(1:100, h);  title('Distribution of distance')
xlabel('Distance from the origin'); ylabel('Number of points')
```

Question 2 (18 points)

A matrix is "symmetric" if it is the same as its transpose. A matrix is "antisymmetric" if it is the negative of its transpose. For example,

$$\begin{bmatrix} 2 & -3 \\ -3 & 5 \end{bmatrix} \text{ is symmetric; } \quad \begin{bmatrix} 0 & -1 & 2 \\ 1 & 0 & 4 \\ -2 & -4 & 0 \end{bmatrix} \text{ is antisymmetric.}$$

A matrix can be both symmetric and antisymmetric; an example is $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. Implement the following function to determine if a *sub*matrix is symmetric and if it is antisymmetric. Do *not* use the transpose operator or the `transpose` function.

Hint: recall the "mirror positions" or "transpose positions" across the main diagonal of a square matrix.

```
function [sym, antisym] = checkSubmatrix(M, L, R)
% Determine if a square submatrix of M is symmetric and if it is antisymmetric.
% M is a non-empty matrix of integer real values.  M may not be square.
% L, R are valid row and column indices of M.  L < R.
% Determine if the submatrix from row L to row R and column L to column R of M
%   is symmetric and if it is antisymmetric.
% sym is 1 (or true) if the submatrix is symmetric; otherwise 0 (false).
% antisym is 1 (or true) if the submatrix is antisymmetric; otherwise 0 (false).
```

Example solutions:

```
% Example soln1: extract sqr submatrix first
A= M(L:R,L:R);
n= size(A,1);
sym= 1;
antisym= 1;
for k= 1:1:n
    for i= 1:1:k
        if A(k, i) ~= A(i,k)
            sym = 0;
        end
        if A(k, i) ~= -A(i,k)
            antisym = 0;
        end
    end
end

% Example soln2: work directly with indices L:R in M
sym= 1;
antisym= 1;
for k= L:R
    for i= L:k
        if M(k,i) ~= M(i,k)
            sym= 0;
        end
        if M(k,i) ~= -M(i,k)
            antisym= 0;
        end
    end
end
```

Question 3 (20 points)

A pixel of an image is said to be "red-dominant" if its red intensity is strictly greater than its green and blue intensities. Implement the following function as specified:

```
function Q = removeRedDominance(P)
% P is a 3-d uint8 array of image data.  P is not empty.
% Q is P with only the red-dominant pixels modified:  each red-dominant
% pixel is assigned a new red intensity that is the average value
% (arithmetic mean) between the green and blue intensities.  The green and
% blue intensities do not change.
```

Example solution:

```
Q= P;

[nr, nc, np]= size(P);

for r= 1:nr
    for c= 1:nc
        if P(r,c,1)>P(r,c,2) && P(r,c,1)>P(r,c,3)
            Q(r,c,1)= P(r,c,2)/2 + P(r,c,3)/2;
        end
    end
end
```

Question 4 (15 points)

Complete the following function as specified:

```
function w = reverseSubvec(v, k, n)
% Reverse a subvector of v, with a maximum subvector length of n, starting
%   at index k
% v: a type char row vector.  v is not empty.
% k: a valid index of v
% n: the maximum length of the subvector to reverse.  n is an integer > 0.
%    If the longest subvector of v starting at index k is shorter than n,
%    then reverse the elements from index k to the end of v.
% w: v with the appropriate subvector reversed
%
% Examples:
%    reverseSubvec('amigo!', 3, 3)  returns  'amogi!'
%    reverseSubvec('amigo!', 3, 5)  returns  'am!ogi'
%    reverseSubvec('amigo!', 2, 1)  returns  'amigo!'
%    reverseSubvec('amigo!', 6, 8)  returns  'amigo!'

w= v;
% Add your code below.  Do NOT use vectorized code.
```

Example solutions:

```
% Example soln1: use accumulator to reverse
bound= min(length(v), k+n-1);
tail= bound;
for head= k:bound
   w(tail)= v(head);
   tail= tail - 1;
end

% Example soln2: compute index of swapped position
bound= min(length(v), k+n-1);
for head= k:bound
    w(head)= v(bound - head + k);
end

% Example soln3: swap elements in subvector
tail= min(length(v), k+n-1);
while k < tail  % loop ends when half-way is reached
    temp= w(k);
    w(k)= w(tail);
    w(tail)= temp;
    k= k+1;
    tail= tail - 1;
end
```

Question 5 (30 points)

The gameboard for a certain game is represented as a matrix G storing type char values. In one step of the game, the gameboard is possibly scrambled by having some row vectors (or subvectors) reversed. The proposed changes to the gameboard G are stored in a 2-d cell array P. Each row of P is a proposed change:

- The first cell stores the index of the row in G to consider changing.
- The second cell stores a search target. If the target is found in that row of G, then reverse the elements in that row of G from the beginning of that row to the end of the *first* target found in that row. Otherwise that row of G should not be changed.

Consider the following example gameboard G and example proposed changes P:

```
G= ['isthereafullmoon'; ...
    'timeforalunytune'; ...
    'mooncakeonaspoon'; ...
    'batbaboonoraloon']
%    1234567890123456
```

```
P= { 2, 'un';        ...
     1, 'mooncake'; ...
     4, 'bat'}
```

Then the updated gameboard given P would be

```
   ['isthereafullmoon'; ...
    'nularofemitytune'; ...
    'mooncakeonaspoon'; ...
    'tabbaboonoraloon'];
```

Observe that the first proposed change, on row 2 of G, was made because the target ’un’ was found. The second proposed change was not made because ’mooncake’ was not found on row 1 of G. The last proposed change was made because ’bat’ was found on row 4 of G.

Implement the function on the following page as specified. For full credit, make effective use of function reverseSubvec from Question 4 (assume it has been implemented correctly). Built-in functions find, strfind, and findstr are forbidden.

Question 5, continued

```
function H = updateBoard(G, P)
% Update gameboard G given the proposed changes in P.
% G: a 2-d simple array of type char representing the gameboard.  G is not empty.
% P: a 2-d cell array with 2 columns.  The first column stores valid and
%    distinct row indices of G.  The second column stores search targets.
%    Each row of P is a proposal to change one row of G.  P has at least
%    one row.
% H: the gameboard updated based on the given rules.  H has the same size
%    and type as G.

% For full credit, make effective use of function reverseSubvec from Question 4.
% Built-in functions find, strfind, and findstr are forbidden.
```

Example solution

```
H= G;
nc= size(G,2);  % number of columns in gameboard
np= size(P,1);  % number of proposals

for rp= 1:np
    % check proposal rp
    rG= P{rp, 1};
    tar= P{rp, 2};  % search target
    nTar= length(tar);
    % Search for target in row rG of char matrix G
    c= 1;
    while  c <= nc-nTar+1  &&  ~strcmp(tar, G(rG, c:c+nTar-1))
        c= c + 1;
    end
    if  c <= nc-nTar+1  % found target, so update G
        H(rG,:)= reverseSub(G(rG,:), 1, c+nTar-1);
    end
end
```

Consider the following example gameboard G and example proposed changes P:

```
G= ['isthereafullmoon'; ...
    'timeforalunytune'; ...
    'mooncakeonaspoon'; ...
    'batbaboonoraloon']
%    1234567890123456
```

```
P= { 2, 'un';        ...
     1, 'mooncake'; ...
     4, 'bat'}
```

Then the updated gameboard given P would be

```
['isthereafullmoon'; ...
 'nularofemitytune'; ...
 'mooncakeonaspoon'; ...
 'tabbaboonoraloon'];
```