Final Exam

Computer Programming for Engineering and Science
Held on 14th of August, 2014 (CSCE 155N, Summer 2014)

Name :

Save : **YES** **NO**

Course No : **CSCE155N Matlab**

Instructions:

1. This is open book, open note, but not open computer or open neighbor. Please do not use email, texting, Morse code, sign language, spy cameras, etc. during the exam.
2. If you have a question about the meaning of an exercise, ask! Getting things wrong because of misunderstandings can be aggravating for me as well as you.
3. If you wish to have the exam saved so that you can retrieve it later, please indicate this. Exams so marked will be saved until at least the second week of classes in the fall.

1. (10 points) Consider the following snippet of code. What do each of the last four lines do and print? Explain. Think carefully, and even though you may not have tried this before, there is a good chance you will get it right!

```
a = 'b';
b = 'c';
c = 14;
eval(a)                          % What happens?
[eval(a), ' = 3 * 7']            % What happens?
eval([eval(a) '= 3 * 7'])        % What happens?
c                                % What happens?
```

2. (10 points) Consider the following functions. What would be the response to the command line **x = nifty(2+3,3) * 4**? Be careful! While this may appear to be the same as an example last summer, there are subtle changes. The concepts, however, still apply.

```
function a = nifty(b,c)
t = c + neato(bummer(c), bummer(b));
a = t - bummer(neato(t,c))
end

function x = neato(y,z)
x = y + z;
end

function x = bummer(y)
x = y * 2;
end
```

3. (10 points) The recursive function for calculating a Fibonacci number as shown in class was horribly inefficient because of repeated calls for the same value. Take a look at the following. What does it display if n is 6? Does it solve the inefficiency problem of the earlier version? Explain.

```
function f = fibonacci(n)
saveF = zeros(1,n);
f = fibo(n)
   function f = fibo(n)     % Note: Nested function so saveF is seen everywhere
   if n < 3
      f = 1                 % Might display here
   elseif saveF(n) > 0
      f = saveF(n)          % Might display here
   else
      f = fibo(n-1) + fibo(n-2)  % Might display here
      saveF(n) = f;
   end
   end
end
```

Here is the original inefficient version:

```
function x = fibo(n)
if n <= 2
   x = 1;
else
   x = fibo(n-2) + fibo(n-1);
end
```

4. (10 points) Consider the following function.

   (a) What would be the response to the call **fun** with no arguments?

   (b) Rewrite it (the function and default x) so that it evaluates infix notation expressions that are formed as nested cell arrays.

```
function t = fun(x)
if nargin == 0
   x = {'+', {'+', 6, {'*', 2, 3}}, {'*', {'+', 5, 1}, 7}};
end

if iscell(x)
   switch x{1}
      case '+'
         t = fun(x{2}) + fun(x{3});
      case '*'
         t = fun(x{2}) * fun(x{3});
   end
else
   t = x;
end
```

5. (10 points) Consider the following code that checks to see if an array is sorted. Create an anonymous function for each of the following cases that responds with **true** if the first argument is bigger than the second. Finally show how to invoke **isSorted** to answer the requests that follow:

   (a) The elements are playing cards as defined in the next exercise. (The numbering of the cards 1 - 52 indicates size.)

   (b) The elements are struct **triangles** with **base** and **height** fields. Size is determined by the areas.

   (c) The elements are struct **student** with fields **age**, **name**, and **GPA**. Size is determined by GPA.

```
function s = isSorted(a, cFun)
   s = true;
   for ii = 2:length(a)
      if ~cFun(a(ii-1), a(ii))
         s = false;
      end
   end
```

6. (10 points) Consider the following code segment. Fill in the bottom **for** loop so that shuffled cards (actually the index of each card) are dealt (in normal rotation sequence) to the four players (represented initially as four empty vectors in cell array **p**).

```
p = {[], [], [], []};
s = {'Clubs','Diamonds','Hearts','Spades'};
for c = 1:52
   card(c).suit = s{ceil(c/13)};
   card(c).val = mod(c,13)+1;
end
x = 1:52;
for ii = 1:100
   c1 = randi([1, 52]);
   c2 = randi([1, 52]);
   t = x(c1);
   x(c1) = x(c2);
   x(c2) = t;
end
for c = 1:52
   %  What goes in here????
end
```

7. (10 points) Consider the following code. Trace through how it works. Effectively, what is its practical purpose?

```
x = [3 4 0 8 9 9 9];
m = length(x);
ii = m;
while ii > 0 && x(ii) == 9
   x(ii) = 0;
   ii = ii - 1;
end
if ii > 0
   x(ii) = x(ii) + 1;
end
```

8. (10 points) Consider the following code. What do the functions **push** and **pop** really do? What is the result of the code that follows the function definitions?

```
function s = push(v, s)
   s = [v s];
end

function [v s] = pop(s)
   v = s(1);
   s = s(2:end);
end

x = '3 4 * 2 5 * +';
stack = '';
for ii = 1:length(x)
   if x(ii) >= '0' && x(ii) <= '9'
      stack = push(x(ii), stack)
   elseif ismember(x(ii), '+*-/'
      [a stack] = pop(stack);
      [b stack] = pop(stack);
      c = eval([a x(ii) b]);
      stack = push(c, stack);
   end
end
[v stack] = pop(stack);
v
```

9. (10 points) Indicate the time complexities of the following examples. They may be $O(\log n)$, $O(n)$, $O(n \log n)$, or $O(n^2)$. Hint: Be especially careful with the last two, because what the claim to be doing might not be what they are actually doing. Do not simply count the number of steps!

```
function b = sort(a)            % What is complexity, including call to small?
for ii = 1:length(a)
   s = small(a(ii:end)) + ii-1;
   t = a(s);
   a(s) = a(ii);
   a(ii) = t;
end
b = a;
```

```
function s = small(a)            % What is complexity of just this function?
s = 1;
for ii = 2:length(a)
   if a(ii) < a(s)
      s = ii;
   end
end
```

```
function exercising(n)          % What is complexity, including call to helfStep?
% My exercise regimen of n steps for each of n days.
for ii = 1:n
   halfSteps(n)
end
```

```
function halfSteps(n)           % What is complexity of just this function?
if n > 1
   fprintf('I have %d steps to go.  Half way is %d steps.\n', n, n/2)
   halfSteps(n/2)
else
   fprintf('Just one more step!\n')
end
```

10. (10 points) Beginning with the following assignment to **a**, what will each of the expressions yield? (First, what is the value of **a**? Hint: **reshape** takes the elements in the order they are stored - column by column - and rebuilds it column by column. We tend to prefer seeing things row by row; that is the reason for the transpose operator (').)

```
a = reshape(1:16, [4, 4])'
```

```
a([1,end],[1,end])
```

```
a([end,1],[end,1])
```

```
a(1:2:3, 1:2:3)
```

```
reshape(a(1:5:16), [2,2])'
```

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	10	
8	10	
9	10	
10	10	
Total:	100	