Midterm Exam

Computer Programming for Engineering and Science
Held on 30th July, 2014 (CSCE 155N, Summer 2014)

Name	:	
Course No	:	**CSCE155N**

Instructions:

1. This is open book, open note, but not open neighbor.
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 answers do not fit in the space provided, use the back of a sheet and very carefully indicate and label this so I don't miss it in grading.
4. You may take the entire period.
5. Cross off any one exercise (on the given page and on the last page) - only do ten of the exercises.
6. Each exercise is worth 10 points, but some may take much more time to complete than others. Go through the exam and do the quick ones first!

1. (10 points) Keeping in mind that a logical array is utilized quite differently from a numeric array, consider the following session in Matlab.

```
>> x = randi(7, 1, 7)
x =
     5     2     3     4     2     6     2

>> [s pos] = sort(x)
s =
     2     2     2     3     4     5     6
pos =
     2     5     7     3     4     1     6

>> x(pos)
ans =
     2     2     2     3     4     5     6

>> gt3 = x > 3
gt3 =
     1     0     0     1     0     1     0

>> x(gt3)
ans =
     5     4     6

>> gt1 = x > 1
gt1 =
     1     1     1     1     1     1     1

>> x(gt1)
ans =
     5     2     3     4     2     6     2

>> int8(gt1)
ans =
    1    1    1    1    1    1    1

>> x(int8(gt1))
ans =
     5     5     5     5     5     5     5
```

(a) What does *s* represent?

(b) What does *pos* represent?

(c) Why do *s* and *x(pos)* give the same vector?

(d) What does the vector of *0*'s and *1*'s in *gt3* mean?

(e) How does *x(gt3)* give just the three larger numbers?

(f) *gt1* and *int8(gt1)* are the same, aren't they? So why are *x(gt1)* and *x(int8(gt1))* different?

2. (10 points) Consider the following code:

```
function p = powr2(n)
   if n == 0
       p = 1;
   elseif mod(n,2)
       p = 2*powr2((n-1)/2)^2;
   else
       p = powr2(n/2)^2;
   end
end
```

(a) If this function is invoked with n being 12, how many times is *powr2* invoked in all (recursively, not including the initial call)? And what is the value of the input argument on each of the invocations? Hint: It is NOT simply 2, the number of appearances in the code.

(b) What is the resulting value when n is 12?

(c) What simple operation is *powr2* performing?

3. (10 points) Consider the following code.

```
>> sort1 = @(e, lst) [lst(lst < e), e, lst(lst > e)];
```

(a) Give the result of *sort1(7, [3 6 9])*

(b) How does the anonymous function *sort1* work?

(c) Give the command that inserts the value v into vector *vec*. Hint: *vec* is needed as an input and is where the result is stored.

4. (10 points) Consider the following code.

```
count = 0;
for a = 1:n
   for b = 1:n
      count = count + 1;
   end
   for b = 1:n^2
      count = count + 1;
   end
   for c = 1:n
      for d = 1:2*n
         for e = 1:n
            count = count+1;
         end
         for f = 1:n
            count = count+1;
         end
      end
   end
end
```

(a) Give a formula (which is a function of n) for the final value of *count*.

(b) How do you know when to add and when to multiply the components to get the formula?

5. (10 points) Consider the following logic expressions.

   (a) Check the expressions in which short circuiting is enabled.

   (b) Circle the expressions in which short circuiting actually occurs.

   (c) Evaluate each expression. hint: The *fprintf* will always be *true* because a non-zero number of characters are being printed.

   (d) What is the side effect of *fprintf*? Point arrows to the instances which actually do the *fprintf*.

   Here are the expressions:

   (a) $25 > 357|fprintf('howdy')$

   (b) $-5 == 8||fprintf('howdy')$

   (c) $45 < 84|fprintf('howdy')$

   (d) $34 < 55||fprintf('howdy')$

   (e) $45 < 245\&fprintf('howdy')$

   (f) $45 < 249\&\&fprintf('howdy')$

   (g) $false\&\&fprintf('howdy')$

   (h) $2 - 16 > 6\&fprintf('howdy')$

6. (10 points) The way you typically see an *fprintf* is with an opening string constant that represents the format for the remaining items. Note that in many instances whether you use 3.14159 (a numeric literal) or *pi* (a numeric variable, though in this case it is predefined) interchangably. This is also true of string literals (in quotes) and string variables. Consider the following interaction in Matlab. Will the final result be as claimed (the message that 8 is positive)? If so, explain how the expression for the formatting in the *fprintf* function works. If not, explain what is wrong.

```
>> msg = char('The number %d is negative.\n', 'The number %d is zero.\n', ...
  'The number %d is positive.\n')
msg =
The number %d is negative.\n
The number %d is zero.\n
The number %d is positive.\n

>> sign(-3)
ans =
    -1

>> sign(0)
ans =
     0

>> sign(52)
ans =
     1

>> fprintf(msg(sign(4*2)+2,:), 4*2)
The number 8 is positive.
```

7. (10 points) What is output by the following fragment of code? Because there is no ending semicolon, note that the last four lines will generate output. Don't miss them! What common operation is simulated by the code?

```
a = [0 5 3 8 7 1 7 7 5];
b = [0 3 5 3 9 3 8 7 3];
s = a+b
c = s > 9
s = mod(s, 10)
s(1:end-1) = s(1:end-1) + c(2:end)
```

8. (10 points) What does the following code print when it is invoked from the command line with *funone*. Be reasonably careful with the spacing. Note that *blanks* is a built-in function which generates a string with the specified number of spaces.

```
function funone()
  a = 0;
  indent(a, 'hello');
  funtwo(a+3);
  indent(a, 'ciao');
  funthree(a+3);
  indent(a, 'done');
end

function funtwo(b)
  indent(b, 'intwo');
  funthree(b+3);
  indent(b, 'outtwo');
end

function funthree(a)
  indent(a, 'inthree')
end

function indent(b, msg)
  fprintf('%d%s %s\n', b, blanks(b),  msg);
end
```

9. (10 points) What is output by the following fragment of code?

```
for a = 4:-1:3
   for b = a:4
      for c = b:-1:a
         fprintf('%1d %1d %1d\n', a, b, c)
      end
      fprintf('in\n')
   end
   fprintf('out\n')
end
```

10. (10 points) Consider the following function definition. Write an expression that uses it (and only it - so no operators!) to calculate the volume of liquid water filling a cylindrical cup that has an ice cube in it, given the radius of the cup as *r*, height as *h*, and the side of the cube as *s*. Remember that only about 9/10 of the ice will be under the surface of the water.

```
function x = mOp(a, b, op)
switch op
   case '+'
      x = a + b;
   case '-'
      x = a - b;
   case '*'
      x = a * b;
   case '/'
      x = a / b;
   case '^'
      x = a ^ b;
   otherwise
      x = 0;
end
```

11. (10 points) Cross off any redundant (unneeded) portions of the code. (The resulting code should always yield the same results, regardless of the input which COULD be outside the specified range.)

```
grade = input('Enter the grade from 0 to 100: ');

if grade > 90 && grade <= 100

    disp('Nice A')

elseif grade > 80 && grade <= 90

    disp('Not a bad B')

elseif grade < 0 || grade > 100

    disp('Invalid input')

elseif grade <= 50 && grade >= 0

     disp('Sorry - you blew it')

elseif grade > 60 && grade <= 70

    disp('Discouraging D')

elseif grade > 50 && grade <= 60

    disp('An E for effort is all you get')

elseif grade > 70 && grade <= 80

    disp('Average C')

end
```

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	10	
8	10	
9	10	
10	10	
11	10	
Total:	100	