typo - 36

Key

Midterm Exam

COMPUTER PROGRAMMING FOR ENGINEERING AND SCIENCE
Held on 29th July, 2015 (CSCE 155N, SUMMER 2015)

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. 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!
6. There are 11 exercises. Only do 10 of them. Clearly cross off the one you do not wish to have counted.

cross off: if you don't, then graders are instructed to cross off your best exercise.

1. (10 points) Complete the following session in Matlab.

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

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

>> x(pos)
ans =
```

1 2 3 4 5 6 7

```
>> x(x)
ans =
```

7 1 6 4 5 2 3

```
>> x == pos
ans =
```

0 0 0 1 1 0 0

```
>> [x(5:7) x(4) x(1:3)]
ans =
```

5 7 2 4 6 3 1

2. (10 points) The *flexSort* function takes two arguments: first is the array to be sorted, and second is a function that is to be used to compare two elements of the array. Examples shown in class are *gt* and *lt* (for greater than and less than).

```
function s = flexSort(a, cFun)
...
   if cFun(a(ii), a(jj))
      t = a(ii);
      a(ii) = a(jj);
      a(jj) = t;
   end
...
end
```

Unfortunately that first argument must be a single dimension array. To allow sorting rows from a two-dimension array, say, where the first column is of lengths and the second column is of widths for an array of rectangles, we would need to make the elements being compared look like *a(ii,:)* to show we want to work with an entire row.

Fortunately there is another solution. We can designate components to variables, so for example, *size* could have two components representing length and width: *size.length* and *size.width*. So let's write a function to plug in for *cFun* that will sort sizes according to the area which is *size.length * size.width* in the case of rectangles.

(a) Write the function *compareArea* that takes two arguments: the size of rectangle 1 and the size of rectangle 2, and returns a true if the size of rectangle 1 is greater than the size of rectangle 2.

(b) Write the command that invokes *flexSort* to sort the array of rectangles named *myRecs*.

```
function greater = compareArea(r1, r2)
   greater = r1.length * r1.width > r2.length * r2.width;
end
myRecs = flexSort(myRecs, @compareArea)
```

3. (10 points) Consider the following code.

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

if missing deduct .5pt

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

(b) Give the result of *sort1(5, [3 5 5 5 6 9 10])* 3 5 5 5 5 5 5 5 6 9 10

(c) How can you fix the function to make is really simply insert in order in all cases?

either change <= to < or >= to >

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
      for c = 1:n
         count = count + 1;
      end
   end
   for b = 1:n
      for c = 1:n
         for d = 1:n
            count = count+1;
         end
         for d = 1:n
            count = count+1;
         end
      end
   end
   for b = 1:n
      count = count + 1;
   end
end
```

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

$2n^4 + n^3 + 2n^2$ (does not need to be simplified)

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

sequence : add

nested : multiply

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*.

   prints 'howdy'

   Here are the expressions:

   (a) $25 < 3 | fprintf('howdy') - 5$ 0

   (b) $-5 == 8 || fprintf('howdy')$ 1

   (c) $45 < 84 | fprintf('howdy')$ 1

   (d) $34 < 55 || fprintf('howdy')$ 1

   (e) $45 < 45 \& fprintf('howdy')$ 0

   (f) $45 < 9 \&\& fprintf('howdy')$ 0

   (g) $true \&\& fprintf('howdy') - 5$ 0

   (h) $2 + 16 > 6 \&\& fprintf('howdy')$ 1

6. (10 points) Consider the following function. If n = pi, t = 9, and r = 5, what is printed? Hint: The ASCII code for '0' is 48, for '1' is 49, etc. The effect of (for example) char(5) + '0' is to add 5 and '0' but to make sure the sum is another ASCII code, not just a number.

π 9 5

```
function pretty(n, t, r)
% n is a double
% t is an integer from 1 to 9
% r is an integer from 1 to 9

f = ['%' char(t)+'0' '.' char(r)+'0' 'f\n']
fprintf(f, n)
```

'%9.5\n'

␣␣3.14159

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?

subtraction

```
x = [0 5 3 8 7 1 7 7 5];
y = [0 3 5 3 9 3 8 7 3];
d = x-y
b = d < 0
d = mod(d, 10)
d(1:end-1) = d(1:end-1) - b(2:end)
```

0 2 -2 5 -2 -2 -1 0 2

0 0 1 0 1 1 1 0 0

0 2 8 5 8 8 9 0 2

0 1 8 4 7 7 9 0 2

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')
  if a <= 12
     funtwo(a+3);
  end
  indent(a, 'outthree')
end

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

0 hello
3 intwo
6 inthree
9 intwo
12 inthree
15 intwo
18 inthree
18 outthree
15 outtwo
12 outthree
9 outtwo
6 outthree
3 outtwo
0 ciao
3 inthree
6 intwo
9 inthree
12 intwo
15 inthree
15 outthree
12 outtwo
9 outthree
6 outtwo
3 outthree
0 done

close enough on spacing is good enough

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

```
for a = 4:-2:2
   for b = a:4
      for c = b:-1:a
         fprintf('%1d %1d %1d\n', a, b, c)
      end
      fprintf('exit c\n')
   end
   fprintf('exit b\n')
end
```

444
exit c
exit b
222
exit c
233
232
exit c
244
243
242
exit c
exit b

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.

$\pi r^2 h - 9/10\, s^3$

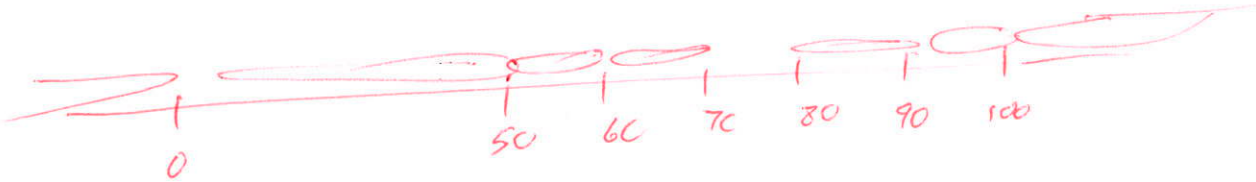
(ugly - multiple correct solutions)

```
function x = mOp(a, op, b)
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
```

mOp(mOp(mOp(T, '*', mOp(mOp(r, '^', 2), '*', h)), '-', mOp(mOp(9, '/', 10), '*', mOp(s, '^', 3))))

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 > 80 && grade <= 90

    disp('Not a bad BB')

elseif grade > 60 && grade <= 70

    disp('Discouraging D')

elseif grade < 0 || grade > 100

    disp('Invalid input')

elseif grade <= 50 && grade >= 0

     disp('Sorry - you blew it')

elseif grade > 50 && grade <= 60

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

elseif grade > 90 && grade <= 100

    disp('Nice A')

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	