Final Exam

Computer Programming for Engineering and Science
(CSCE 150E, Spring 2010)

Held on 3rd and 6th May, 2010 *3 and 6 May, 2010*

Name :

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

Course No : **CSCE150E Matlab**

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. This exam is being given on two days. You are on your honor not to reveal anything to members coming the later date (assuming you are attending the first date.)
4. There are 120 points in this exam. You are responsible for doing all the exercises. However, look at the score card on the last page. You will notice a column labeled *Revised* with a total of 100 points. You may revise downward the number of points of any or all exercises to as low as a third the original (rounding down), so that the total is indeed 100. For example, you could reduce the points for exercises 8, 9, 10, and 11 down to 5 each. That would reduce the points by 20 which drops the total to 100. If you do not indicate any revisions, or if your revisions do not result in a total of 100, the original points will be used and your total scaled by 5/6.
5. 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 the second week of classes in the fall.

1. (10 points) The function *randperm(5)* returns a random permutation (scrambling) of the numbers 1 through 5. The function *sort(a)* returns two arrays: first, the sorted array *a*, and second, the positions of each of the sorted elements in the original array *a*. Examples:

```
>> randperm(5)
ans  =
     3     5     2     1     4
>> [x y] = sort([31 51 22 593 5 -5])
x =
    -5     5    22    31    51   593
y =
     6     5     3     1     2     4
```

   Assume Joe, Bill, Sammy, Mark, and Jim scored 62, 37, 99, 100, and 16 respectively on an exam. You may wish to take advantage of the features discussed above.

   (a) Form the names into a cell array. Form the grades into a regular array. Sort the grades in ascending order. Use the result to sort the names in ascending order by grade.

   (b) Scramble the original cell array of names into a new random permutation.

2. (10 points) The function $max([52, 36, 17, 99, 5])$ returns two values: first, the value of the largest element (in this case it is 99), and second, the position of the largest element (in this case it is 4). Recall that arrays can be torn apart and rebuilt easily with Matlab. Recall that the opening line of a function that returns two results would look something like this: $function[x, y] = funny(a)$. Both x and y would presumably be assgned values somewhere in the function.

   Let array a consist of 52, 36, 17, 99, and 5.

   (a) What is the result of $b = a([1 : 2, 4 : end])$?

   (b) What is the result of $b = [b, a(3)]$ (using the result of the previous part)?

   (c) Write a function named *extract* containing one argument (an array) that extracts the largest element and returns both the extracted element and the remaining array (as two separate results).

   (d) What would be a better name for the following function?

```
function y = whatdoido(x)
y = [];
for ii = 1:length(x)
    [m x] = extract(x);
    y = [m y];
end
```

3. (20 points) A 2-D array named *pic* contains numbers in the range of 0 to 10. These represent brightness (10 being brightest) at points in a photo taken of the gulf off Louisiana. The oil is highly reflective, so brightness of oil covered areas is 8 or above. The following steps illustrate a simplified scenario that simulates the movement of the oil. In reality, wind would be a vector field, water currents would be another vector field, rate of spread would depend on a number of factors, etc., but all of these are easily handled by Matlab, and I hope suggest to you the importance of computation.

   (a) Give a command(s) that will yield an array containing *true* for those locations likely covered with oil.

   (b) Give a command(s) that will calculate the area (number of elements that are true) of the oil covered water. (You may use the results of part (a) here and in the following part.)

   (c) Give a command(s) that will report how far north the oil has spread (the row with the smallest index containing a *true*).

   (d) Assuming oil spreads outward in all directions at the same speed, give a command(s) that show the extent of the oil after one unit of time (the time it takes to spread from one sampling point to the next).

   (e) Wind also affects the movement of the oil. Give the command(s) that take the result of the previous part, and show the affect if the wind blows this mess one unit to the east (right).

   (f) The well head continues to emit oil, so give the command(s) that force a point representing the location of the well head (assume this is $(rWell, cWell)$) continues to show oil.

4. (10 points) Consider the following function definition. Now picture yourself at the command window, needing to use only that function to calculate the polynomial $2x^2 + 5xy - 3y^2$.

   (a) What do you need to type to do this using just one command?

   (b) Repeat, assuming the arguments to multiOp are in the order *a*, *op*, *b*.

   (c) Repeat, assuming the arguments to multiOp are in the order *op*, *a*, and *b*.

   (d) Using the answers above as a guide, express the polynomial using prefix, infix, and postfix notation, respectively.

```
function x = multiOp(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;
end
```

5. (10 points) The way you typically see an *fprintf* is with an opening string literal that represents the format for the following items. Note that in many instances you can 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.

   (a) What is printed by the code (below)?

   (b) Modify the *fprintf* line so that it doesn't look so weird (in other words, the first argument is the more familiar string literal.)

   (c) Rewrite the code so it uses a *while* loop instead of a *for* loop.

```
f = {'one','two','three','six'}
for ii = 1:10
   d2 = mod(ii,2) == 0);
   d3 = 2 * (mod(ii,3) == 0);
   fprintf(['The number %d is divisible by ' f{1+d2+d3}], ii);
end
```

6. (10 points) Consider the following code, which is very similar to what you have seen a couple times already!

   (a) Cross off any redundant (unneeded) portions of the code. (The resulting code should always yield the same results.)

   (b) Using division by 10 and a correct choice of *ceil* or *floor*, rewrite the code to consist of a *switch* structure, requiring only a single test value for each *case*.

```
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 <= 50 && grade >= 0
    disp('Sorry - you blew it')
elseif grade < 0 || grade > 100
    disp('Invalid input')
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
```

7. (10 points) Consider the following code.

   (a) Give an algabraic formula for the final value of *count*.

   (b) How deeply nested are the loops?

   (c) What is the relationship between the (highest) exponent of x in the formula requested above and the level of loop nesting?

```
count = 0;
for a = 1:x
   for b = 1:x
      for c = 1:x
         count = count + 1;
      end
      for d = 1:x
         count = count + 1;
      end
   end
   for e = 1:x
      count = count + 1;
   end
end
```

8. (10 points) Consider the following code.

   (a) 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.

   (b) Unlike the previous exam, not all the arguments have the same name. Does this make any difference? Explain.

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

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

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

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

9. (10 points)

   (a) Replace *funthree* of the previous exercise with the following version. Now what is printed?

   (b) A new workspace is allocated each time a function is called, and thrown away when the function is exited. At the maximum, how many workspaces are in existance at any given moment.

```
function funthree(n)
  indent(n, 'inthree');
  if n < 9
    funtwo(n+3);
  end
  indent(n, 'outthree');
end
```

10. (10 points) Consider the following structure.

    (a) Rewrite the code for the structure in exactly one command.

    (b) Write a function *dist3* that accepts two points (in 3-D) as arguments and returns the distance between them.

    (c) Two triangles are called 'similar' if the lengths of the corresponding sides are all proportional. Noting that in our structure, the ordering of points is arbitrary, write a function *similar* that yields *true* if and only if two triangles (the arguments) are similar.

```
point3(3) = struct('x', 1, 'y', 3, 'z', 5);
point3(2) = struct('x', 7, 'y', 9, 'z', 13);
point3(1) = struct('x', 3, 'y', 5, 'z', 8);
triangle = struct('color', 'red', 'coordinates', point3);
```

11. (10 points) Answer the following.

    (a) What is the basic idea of event driven programming?

    (b) What is the basic idea of GUI?

    (c) Does it make sense to have event driven programming without a GUI? Explain.

    (d) What is the basic idea of a call back function?

    (e) What is a handle, and how does a programmer work with it?

    (f) What practical service is provided by GUIDE when it constructs the *handles* argument?

    (g) What clues would lead you to suspect that *handles* is a struct?

Question	Points	Revised	Score
1	10		
2	10		
3	20		
4	10		
5	10		
6	10		
7	10		
8	10		
9	10		
10	10		
11	10		
Total:	120	100	