Final Exam

COMPUTER PROGRAMMING FOR ENGINEERING AND SCIENCE
Held on 27th of April, 2012 (CSCE 155N, SPRING 2012)

Save? : YES NO

Name : Key

Course No : **CSCE155N Matlab**

Instructions:

1. Please indicate above your name whether of not you with the exam to be saved for you to pick up later. If you wish, I will hold exams through the first couple weeks of fall semester. After that they will be discarded.
2. This is open book, open note, but not open neighbor. Please do not use email, texting, sign language, etc. during the exam.
3. 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.

1. (15 points - 3 points for each part) Consider the code on the next page. Assume ASCII file *listG.dat* contains a list of random integers from 0 to 100, one per line.

   (a) Describe what gets written into lists A through F in stage 1. Where would a 35 go? a 57 go? a 90 go? a 99 go?

   Numbers according to corresponding letter grade
   35 → listF.dat
   57 → F
   90 → B
   99 → A

   (b) What is the purpose of the arithmetic expression in the switch statement?

   scale grades by factor of ten, force to integer, shift by 5 so that switch applies easily

   (c) Describe what gets written back into the original list in stage 2. Is there something changed in the ordering of the contents?

   all A grades, then B grades, etc. pushed back in to original file. So order is now by A, B, C, D, F, though grades within a grade level retain original order

   (d) What is the purpose of *fname* which keeps changing and *gs*?

   allows easy looping by systematically going through A, B, C etc. in the 5th slot of listx.dat, generating a legitimate file name

   (e) Is there any flaw in the logic and construction of the code? If so, describe. (Ignore typos that may have gotten past me!)

   not that I know of!

```
fidg = fopen('listG.dat');        % beginning of stage 1
fidA = fopen('listA.dat','w');
fidB = fopen('listB.dat','w');
fidC = fopen('listC.dat','w');
fidD = fopen('listD.dat','w');
fidF = fopen('listF.dat','w');
while ~feof(fidg)
   g = str2num(fgetl(fidg));
   switch max(1, ceil(g/10) - 5)
      case 1
         fprintf(fidF, '%d\n', g);
      case 2
         fprintf(fidD, '%d\n', g);
      case 3
         fprintf(fidC, '%d\n', g);
      case 4
         fprintf(fidB, '%d\n', g);
      case 5
         fprintf(fidA, '%d\n', g);
   end
end
fclose(fidg);
fclose(fidA);
fclose(fidB);
fclose(fidC);
fclose(fidD);
fclose(fidF);
fidg = fopen('listG.dat','w');      % beginning of stage 2
fname = 'listx.dat';
gs = 'ABCDF';
for ii = 1:5
   fname(5) = gs(ii);
   fid = fopen(fname);
   while ~feof(fid)
      fprintf(fidg, '%s\n', fgetl(fid));
   end
   fclose(fid)
end
fclose(fidg);
```

2. (10 points) Begin with an array *days* containing the number of days in each (non-leapyear) month (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31). Using a *for* loop and an assignment of the form *totDays(?:?) = totDays(?:?) + days(?:?)*, the array *totDays* should end up with the cumulative number of days for the year at the end of each month (31, 59, 90, 120, 151, etc.)

```
totDays = zeros(1,12);
days = [31 28 31 30 31 30 31 31 30 31 30 31];
for m = 1:12
    totDays(m:end) = totDays(m:end) + days(1:13-m);
end
```

3. (10 points) Write a function *isomorphic* that takes two *triangles* as arguments and returns *true* if they are isomorphic (the same length sides and angles) and *false* if they are not. A triangle is composed of three *points*. A point is composed of an x and a y coordinate. Note that you do not need to know the angles. Consider the the sides in each triangle may be in arbitrary order, so you may need to test all pairings.

```
function iso = isomorphic(t1, t2)
    e1(1) = dist(t1.point(1), t1.point(2));
    e1(2) = dist(t1.point(2), t1.point(3));
    e1(3) = dist(t1.point(3), t1.point(1));
    e2(1) = dist(t2.point(1), t2.point(2));
    e2(2) = dist(t2.point(2), t2.point(3));
    e2(3) = dist(t2.point(3), t2.point(1));
    e1 = sort(e1);
    e2 = sort(e2);
    iso = all(e1==e2)    % better to allow small variance
                         because value are real
```

```
function d = dist(p1, p2)
    dx = p1.x - p2.x;
    dy = p1.y - p2.y;
    d = (dx^2 + dy^2)^0.5;
end
end
```

- + 1 2 * + 3 * 4 5 6

4. (10 points) Show each of the following as binary parse trees. Then convert into the missing two forms (so we get prefix, infix, and postfix for each). Note that each number is one digit! So treat 456 as a 4, a 5, and a 6.

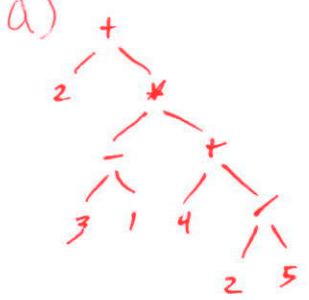
   (a) $2 + (3 - 1) * (4 + 2/5)$
   (b) $- + 12 * +3 * 456$
   (c) $123 - + 45 - 6 + *$

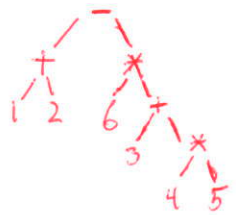
a) + 2 * - 3 1 + 4 / 2 5 (pre)
2 3 1 - 4 2 5 / + * + (post)

b) (1+2) - (3+((4+5) * 6)) (in)

(1+2) - 6(4*5+3) (in)
12+ 6345*+*- (post)

c) (2-3+1)*(4-5+6) (in)
*+1-23+6-45 (pre)





5. (~~10~~ 5 points) Consider the following code. Rearrange only the second part to maximize the number of needed comparisons. This is in contrast to earlier exams! (There is more than one solution.)

```
grade = input('Enter the grade from 0 to 100: ');
~~elseif~~ grade < 0 || grade > 100
    disp('Invalid input')

elseif grade > 80 && grade <= 90          % second part
    disp('Not a bad B')
elseif grade <= 50 && grade >= 0
    disp('Sorry - you blew it')
elseif grade > 90 && grade <= 100
    disp('Nice A')
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
```

1 2 3 4 5 6

B	E
C	D
D	C
E	B
F	A
A	F

endpoints last, rest largely interchangeable

-1 for each redundant statement that could have been required

6. (5 points) Consider the following code. What is the final value of *count*?

```
count = 0;
for a = 1:n
    for b = 1:n
        count = count + 2;
    end
    for c = 1:n
        for d = 1:n
            count = count + 3;
        end
        for e = 1:n
            count = count + 1;
        end
    end
    for f = 1:n
        count = count - 1;
    end
end
```

$4n^3 + n^2$

7. (15 points) Create a GUI element *hit* and array of 5 *moles* along with callback functions that plays a simple "Whack-a-Mole" game. Poking *hit* generates a random number from 1 to 5 that is used to turn the associated mole's string to "hit me!". Use *tic* and *toc* to determine the time it takes to hit the mole. After being hit, the string changes to "ouch!".

8. (10 points) In terms of return values and side effects, compare/contrast the *fprintf* and *sprintf* functions. When is each useful?

Side effect [sprintf – o/p of the assignment is suppressed
fprintf – string displayed

Return value [sprintf – string printed
fprintf – number of characters in string

9. (10 points) What is printed when the following function is invoked with *curse(3)*? How about with *curse(4)*?

```
function curse(n)
   for ii = 1:n
      fprintf('A %d %d\n', n, ii)
      curse(n/2)
      fprintf('B %d %d\n', n, ii)
   end
```

curse (3)

A 3 1
A 1 1
B 1 1
B 3 1
A 3 2
A 1 1
B 1 1
B 3 2
A 3 3
A 1 1
B 1 1
B 3 3

curse (4)

A 4 1
A 2 1
A 1 1
B 1 1
B 2 1
B 4 1
A 4 2
A 2 2
A 1 2
B 1 2
= So on

10. (10 points) Recognizing that for most students, this Matlab course is ancillary to the main area of study, what benefits (if any) do you see in the course for yourself in your chosen major?

Anything

Question	Points	Score
1	15	
2	10	
3	10	
4	10	
5	5	
6	5	
7	15	
8	10	
9	10	
10	10	
Total:	100	