Instructions:

- This document (and thus your scanned submission) should consist of 10 pages. You may attach additional pages if your work does not fit on those provided.
- This is an open-note exam and is designed to take approximately 150 minutes (2.5 hours) to complete. No collaboration is allowed.
- The exam is worth a total of 150 points.
- Read each problem completely, including any provided code, before starting it.
- If a question is unclear, e-mail Dr. Muhlberger; do not ask anyone else. Check Canvas before submitting your exam in case we need to announce any clarifications to the whole class.
- Clarity, conciseness, and good programming style count for credit. That being said, function header comments are not required in an exam setting.
- Indicate your final answer. If you supply multiple answers, you may receive a *zero*.
- Use only MATLAB code. No credit for code written in other programming languages.
- Assume there will be no input errors.
- Vectorized code is not required (but it may make things easier to write).
- You may write a subfunction as part of your solution if you think it will clarify your code.
- Do not use `switch`, `try`, `catch`, `break`, or `continue` statements.
- **Do not use built-in functions that have not been discussed in the course.** Limit yourself to the following MATLAB predefined functions: `abs`, `sqrt`, `sin`, `cos`, `log`, `rem`, `floor`, `ceil`, `round`, `min`, `max`, `sum`, `rand`, `zeros`, `ones`, `linspace`, `length`, `size`, `isempty`, `transpose`, `strcmp`, `str2double`, `double`, `uint8`, `char`, `cell`, `sort`, `fopen`, `fgetl`, `feof`, `fclose`, `input`, `fprintf`, `sprintf`, `disp`, `plot`, `fill`, `nargin`, `error`

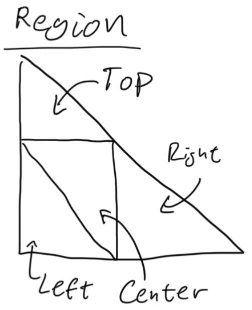
Examples: `sin(pi/6)` → 0.5, sine of argument (in radians)
`rem(5,2)` → 1, the remainder of 5 divided by 2
`max(-4,3)` → 3, largest argument
`sum([0 4; 1 -1])` → [1 3], vector of column sums of the *matrix* argument
`rand()` → a random real value in the interval (0,1)
`log(1)` → 0, natural logarithm
`floor(6.9)`, `floor(6)` → 6, rounds down to the nearest integer
`zeros(1,4)` → 1 row 4 columns of zeros
`length([2 4 8])` → 3, length of a vector
`[nr,nc,np]=size(M)` → dimensions of `M`: `nr` rows, `nc` columns, `np` layers
`plot([3 1],[-5 0],'r')` → draws a line from (3,-5) to (1,0) in red
`[y,idx]=sort(x)` → elements of `x` sorted in ascending order returned in `y` with the property that `y(k)=x(idx(k))`

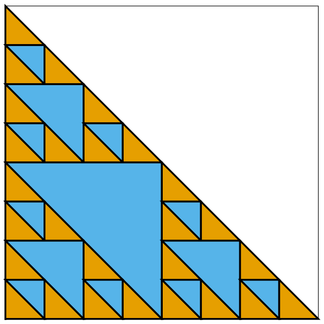
Question 1. (17 points)

(a) Consider the following pseudocode for a recursive function that draws a figure like the one below (drawn to level 3); note: a "region" is a triangle that may be divided into four sub-triangles:

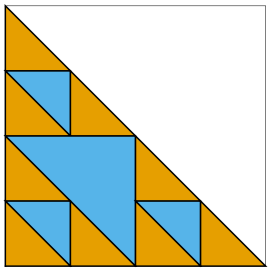
```
Function fillFractal(region, level):

If level is 0:
    Fill whole region orange
Otherwise:
    fillFractal(top triangle, level-1)
    Fill center triangle blue
    fillFractal(right triangle, level-1)
    fillFractal(left triangle, level-1)
```

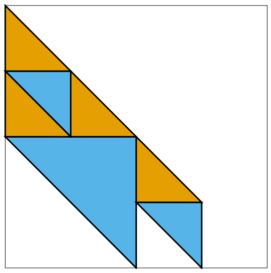




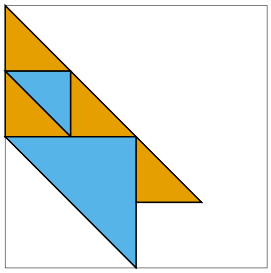
If invoked at level 2, which will be the appearance of the graphics produced immediately after 5 calls to `fillFractal()` have returned?



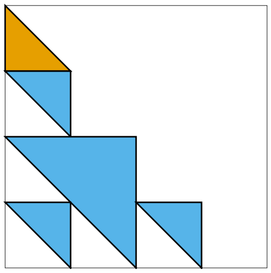
A



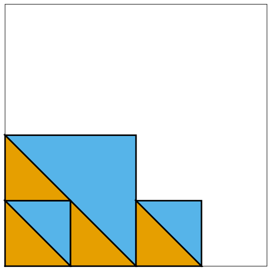
B



C



D



E

Answer: C

(b) Consider a class `Fastener` which has one protected property, `diameter`, and one private property, `length`. Then consider a class `Screw` which inherits from `Fastener` and has a private property `pitch`. `Fastener` has a public method `getLength()` which returns its length property, and the developer is trying to add a public method `getVolume()` to `Screw` to compute the volume of wood displaced by a screw.

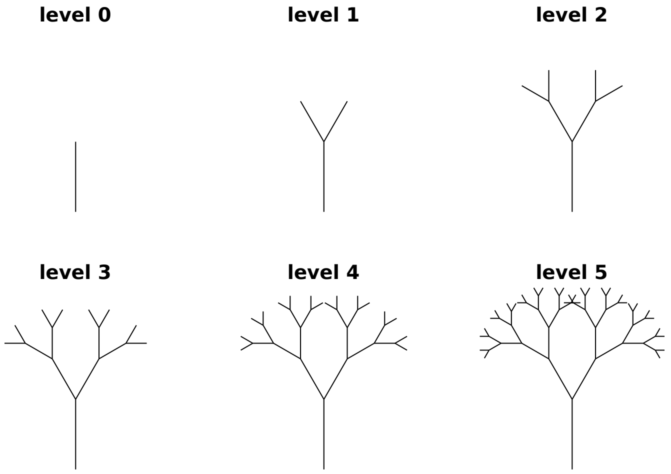
Which of the following statements are true?

A. If `s` is a handle to a `Screw` object in my script, then I can call `s.getLength()`.
B. If `s` is a handle to a `Screw` object in my script, then I can access `s.diameter`.
C. Inside the `Screw` classdef, the `getVolume()` method can access `self.diameter` (where `self` is the first parameter of the method).
D. Inside the `Screw` classdef, the `getVolume()` method can query its length, diameter, and pitch (by reading its properties and/or invoking its other methods).
E. If `fs` is a (non-cell) array of `Fastener`s, then I can invoke `getVolume()` on elements of `fs`.
F. The `length` property does not exist in objects of class `Screw`.

True statements: A, C, D

Question 2. (21 points)

(a) Implement the function `drawTree()` below to recursively draw a tree to a specified "level" (see figure). The branches in each level are 2/3 as long as the branches in the the previous level, and each pair of branches sharing a starting point are separated by 60° (±30° from the direction the previous level's branch was drawn in).



Hint: Remember that if you move a distance r at an angle θ from the x-axis, then your x coordinate will change by $r\cos\theta$ and your y coordinate will change by $r\sin\theta$.

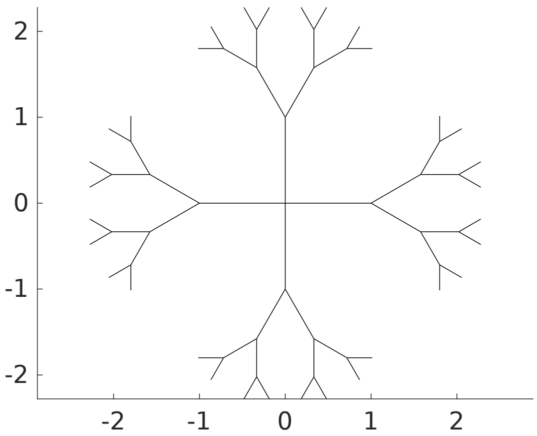
```
function drawTree(x, y, d, theta, level)
% Draw a tree to a specified "level".
% The tree's stem is drawn as a black line of length `d`, starting from the
% point (x,y) and extending in the direction `theta` (measured in radians
% counter-clockwise from the x-axis).  The next level of the tree (if any)
% should branch from the end of the stem and be composed of stems 2/3 as
% long and separated from each other by 60 degrees.
```

```
xf = x + d*cos(theta);
yf = y + d*sin(theta);
plot([x, xf], [y, yf], 'k')
if level > 0
    drawTree(xf, yf, 2*d/3, theta + pi/6, level-1)
    drawTree(xf, yf, 2*d/3, theta - pi/6, level-1)
end
```

(b) Complete the following script to draw the figure shown, making use of the `drawTree()` function from part (a):

```
% Draw 4 trees in a "plus" arrangement
figure
hold on
axis equal
```

```
drawTree(0,0,1,0,3)
drawTree(0,0,1,pi/2,3)
drawTree(0,0,1,pi,3)
drawTree(0,0,1,3*pi/2,3)
```



Question 3. (18 points)

Consider a `Student` object with the following public methods (only the headers are shown):

```
function s = getName(self)
% Return the name of the student referenced by `self`.

function s = getScore(self)
% Return the final exam score of the student referenced by `self`.
```

(a) Assume you are given a 1D cell array containing handles to `Student` objects. Write a free function `filterStudents()` that extracts the names of students whose final exam scores fall in the range `sLo` $\leq$ *score* $<$ `sHi` (where `sLo` and `sHi` are input parameters of the function and *score* is the student's exam score). The function should accept the cell array of students as an additional input and return a 1D cell array of character vectors. The students in the input cell array are unordered, and you should *not* attempt to sort them as part of your solution.

```
function names = filterStudents(sLo, sHi, students)
names = {};
for k = 1:length(students)
    s = students{k}.getScore();
    if s >= sLo && s < sHi
        names = [names, students{k}.getName()];
    end
end
```

(b) If the input cell array contained 1000 students, what are the minimum and maximum number of comparisons your function might make to perform this task?

Minimum number of comparisons: 1000

Maximum number of comparisons: 2000

Question 4. (22 points)

(a) [This is a continuation of the scenario described in question 3] Assume now that the cell array of students is already sorted by score in ascending order. Complete the following function to find the index of the *first* student whose score is *at least* x using a binary search strategy. Only write expressions in the three blanks; do not add or modify any other code.

```
function k = scoreSearch(students, x)
% Return the index `k` of the first student in `students` whose score is at
% least as large as x.  `students` is a 1D cell array of handles to Student
% objects, sorted in ascending order by their scores.  If no students have
% a score >= x, then `k` will be 1 larger than the number of students.
```

```
lb = 1;                     % Smallest possible index of target
k = length(students) + 1;   % Largest possible index of target

while ( lb < k )                          % <--
    m = floor((lb + k)/2);

    if ( students{m}.getScore() < x )     % <--
        lb = m + 1;
    else

        k = ( m );                        % <--
    end
end
```

(b) Write a function filterSortedStudents() that makes effective use of scoreSearch() to perform the same task as filterStudents() when the input is assumed to already be sorted by score. Avoid unnecessary score comparisons.

```
function names = filterSortedStudents(sLo, sHi, students)
kBegin = scoreSearch(students, sLo);
kEnd = scoreSearch(students, sHi);
names = {};
for k = kBegin:(kEnd-1)
    names = [names, students{k}.getName()];
end
```

(c) If the input cell array contained 1000 students, what is the approximate number of score comparisons that filterSortedStudents() would have to perform in the worst case?

A: 10   B: 20   C: 500   D: 1000   E: 2000   Answer: B

Question 5. (16 points)

Assume there exists a function compare(a, b) that, given two character vectors, returns 1 if the thing described by a is “better than” the thing described by b, returns -1 if b is “better than” a, and returns 0 if the two things are of equal value. For example, compare('reindeer', 'people') would return 1 if reindeer are better than people. The author of this function takes pride in their consistency and guarantees that the comparisons respect a total ordering (in other words, you’ll never find something that is both better than reindeer and worse than people, so it’s possible to sort a list using this ranking).

We would like to sort a cell array of strings according to this ranking using the *bubble sort* algorithm (see comments). Complete the following function as specified in order to sort the array in *descending* order (best thing first, worst thing last); note that the bubble procedure is inlined. Add your code beneath the “TODO” comment block, leaving the surrounding code unchanged.

```
function words = bubbleSortTheThings(words)
% Sort words from "best" to "worst" using the `compare()` function to
% rank the things represented by the words.  `words` is a cell array
% of character vectors.  The sorting procedure is stable (does not
% reorder things of equal value).

n = length(words);
k = 1;
didSwap = 1;
while k < n && didSwap
    didSwap = 0;

    % TODO: Find the best element in words(k:n) and bubble it up to
    % words{k}.  Do this by traversing the subarray from right to left
    % and swapping an element with its left neighbor if they are in
    % ascending order.  If any swaps are performed, set didSwap to 1.

    for i = n:-1:(k+1)  % Iterate right-to-left
        if compare(words{i-1}, words{i}) == -1
            tmp = words{i};
            words{i} = words{i-1};
            words{i-1} = tmp;
            didSwap = 1;
        end
    end

    k = k + 1;
end
```

Question 6. (26 points)

Let F be a 1D (non-cell) array of Film object handles. A Film object has these public properties:

- title – the title of the film (a char row vector)
- country – the listed country of origin of the film (a char row vector)
- duration – the movie length in minutes (a type double scalar)

All the films represented in F are different and are to be reviewed by volunteer jurors of a film festival. For the Feature Film Award, a juror will be asked to review exactly three films, each at least 40 minutes, totaling to no more than six hours, and include at least two countries. Write a code snippet below to find all unique combinations of three films that are possible for a Feature Film Award juror to review given F (the array of Film handles) and the criteria specified above. Store each unique combination of three films that meets the criteria in one row of a m-by-3 cell array C, one film title in each cell of the row. In this notation, m is the number of unique combinations found; if no three films meet the criteria, then m is 0 (i.e., C is an empty cell array).

For full credit, your code should be efficient by avoiding *unnecessary* iteration. That being said, do not attempt to sort F as part of your solution.

```
% Assume `F` is a non-empty array of Film handles as described above.
% Write your code below.
```

```
% Exlcude short films (and too long ones) once and for all (efficiency)
i= 0;
for k= 1:length(F)
    if F(k).duration >= 40 && F(k).duration <= 360
    % OK if student doesn't check <=360
        i= i+1;
        FF(i)= F(k);
    end
end

% Find unique combos
C= {};
for i= 1:length(FF)-2  % going to length(FF) is ok
    for j= i+1:length(FF)-1  % going to length(FF) is ok
        ijtotal= FF(i).duration + FF(j).duration;
        % no penalty if student checks total time in innermost loop
        if ijtotal <= 360
            for k= j+1:length(FF)
                total= ijtotal + FF(k).duration;
                if total <= 360 && (...
                        ~strcmp(FF(i).country,FF(j).country) || ...
                        ~strcmp(FF(i).country,FF(k).country) || ...
                        ~strcmp(FF(j).country,FF(k).country) )
                    C= [C; {FF(i).title, FF(j).title, FF(k).title}];
                end
            end
        end
    end
end
```

Question 7. (30 points)

Consider the relationships between characters in a fictional story. If each character has an ID number, then we can represent friendships between characters at a certain point in the plot using a square numeric matrix M, where M(i,j) is 1 if the characters with IDs i and j are friends (otherwise it is 0). Assume that all friendships are mutual (that is, if Alice is friends with Bob, then Bob is also friends with Alice).

(a) Write a function addFriendship() to add a new friendship to a relationship matrix. The function should take 3 arguments: the current relationship matrix and the IDs of the two characters who became friends. It should return the updated matrix.

```
function M = addFriendship(M, c1, c2)
M(c1,c2) = 1;
M(c2,c1) = 1;
```

(b) The mapping between characters and their IDs is captured by a cell array of char vectors, e.g. names, where names{k} is the name of the character with ID k. Write a function mostFriends() that determines who has the most friends. It should accept two arguments: a relationship matrix and a cell array of names; and it should return two values: a char array containing the name of the character with the most friends, and a numeric scalar counting how many friends they have. If multiple characters are tied for the most friends, the name of the one with the smaller ID should be returned.

```
function [name,count] = mostFriends(names, M)
count = 0;
for r = 1:length(names)
    mycount = 0;
    for c = 1:length(names)
        mycount = mycount + M(r,c);
    end
    if mycount > count
        count = mycount;
        name = names{r};
    end
end
```

(c) Implement the following function to print out a character's social network; you may assume that the diagonal of the relationship matrix is all zeros.

```
function showFriendNetwork(names, M, c)
% Print the friends network of the character with ID number `c`.
% First, print a header line specifying the name of character `c`.
% Then, for each friend of `c`, print a line starting with their name,
% followed by a colon, followed by the names of their friends separated by
% commas (but not including `c`).  `names` is a 1D cell array of character
% names (indexed by their ID), and `M` is a 2D array such that `M(i,j)` is
% 1 if character `i` is friends with character `j`.
% Example output:
%     Friends network for Cloud
%     Barrett: Biggs, Wedge, Tifa
%     Aerith:
%     Tifa: Barrett
```

```
fprintf('Friends network for %s\n', names{c})
n = size(M,1);
for k = 1:n
    if M(k,c) == 1
        fprintf('%s: ', names{k})
        hasFriends = false;
        for cc = 1:n
            if M(k,cc) == 1 && cc ~= c
                if hasFriends
                    fprintf(', ')
                end
                fprintf(names{cc})
                hasFriends = true;
            end
        end
        fprintf('\n')
    end
end
```