CS1112 Fall 2010 Project 6 Part 2 **due Thursday 12/2 at 11pm**

(Part 1 appears in a separate document. Both parts have the same submission deadline.)

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the project you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group.

Objectives

Completing Part 2 of this project will improve your understanding of user interactive graphics, connectivity matrices, and recursive functions. You will be able to see how a simple diagram can become an intricate structure through recursion.

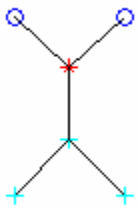
2 Tree Diagrams

You will write a set of functions that allows a user to draw a shape called a "branch" and then copy the branch repeatedly to create a more complex picture, or a "tree." The branch is drawn by placing and connecting *n* points where *n* is a function parameter (*n* >= 3). The diagram below on the left shows an example branch with *n* = 6.

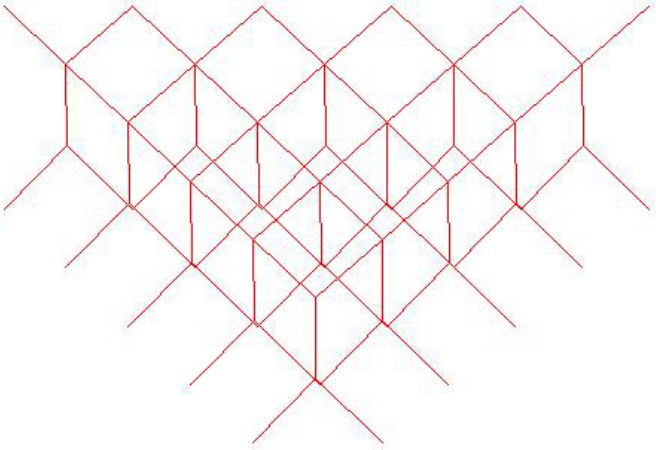
Once the branch design is complete, it will be copied *recursively* to form a "tree" such as the image below on the right. Take a closer look—you should be able to identify the individual branches that make up the tree.

Begin by downloading three files, `drawBranch.m`, `drawTree.m`, and `drawTreeGraphics.m` from the course website. The script `drawTreeGraphics` calls the functions `drawBranch` and `drawTree`, which you must complete, to draw recursive trees such as the one shown below.

Branch



Tree



2.1 Drawing a Branch

2.1.1 Selecting n Points

The first step is to allow the user to create and display a branch that consists of *n* connected points. The function `drawBranch` is specified below and will handle this; it outputs vectors `x` and `y`, and a connectivity matrix `C`.

```
function [x, y, C] = drawBranch(n)
% Create a branch of up to n connected points.
% First point is starting point at origin, marked by a red asterisk.
% The next set of points are user input, marked by cyan '+'s.
% The last two points are recursive connection points, marked by blue 'o's.
% x and y are position vectors: the kth point is located at (x(k),y(k)).
% Connectivity matrix C is symmetric and C(r,c) is 1 if point r and point c
%   are connected; otherwise it is 0.
```

The first point is fixed at the origin (0,0) and is marked by a red asterisk (*). The other `n`-1 points are determined by user clicks. Use built-in function `ginput`, e.g., the statement `[a,b]= ginput(1)` assigns to variables `a` and `b` the x- and y-coordinates, respectively, of a user mouse click.

The user must place these points in a square of side length 2 centered at the origin. Make sure that your code rejects any point clicked outside of the acceptable area (keep prompting the user until an acceptable click is made). All but the last two user-selected points should be marked by cyan pluses (`+`). The last two user-selected points are called "recursive connection points" and should be marked by blue circles (`o`). The difference between the regular and recursive connection points will be important when drawing the tree in §2.2, to be explained later.

Thus, the `n` points look like this:

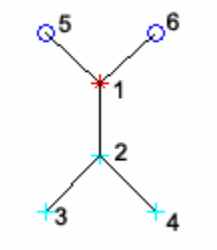
Point	Type	Marking
1	starting point fixed at (0,0)	red *
2	user input regular point	cyan +
⋮	⋮	⋮
n-2	user input regular point	cyan +
n-1	user input recursive connection point	blue o
n	user input recursive connection point	blue o

`drawBranch` should display the points as specified in the figure window after each user click. Give appropriate instructions for the user to follow in the title of the figure, e.g., a message such as "click regular point 2," "click recursive connection point 6," etc.

2.1.2 Connecting the Dots

Now that all of the points have been placed, the user can select pairs of points to connect them. Prompt the user to click and select one of the existing points using the title area of the figure. A point is selected if the click is within a certain distance of the point, which you should define as something reasonable. If the click is not near any points, prompt the user to try again. If a user click does select a point, prompt the user to click a different point with which to form a connection. Once two different points are properly selected, draw a black line to connect them. This process repeats until the user clicks outside of the drawing box to indicate that there are no more connections.

The connectivity matrix, `C`, indicates which points are connected. So if `C(r,c)` is 1, then point r and point c are connected. The matrix is symmetric, so `C(c,r)` will have the same value as `C(r,c)`. Also, a point is considered to be connected to itself so the diagonal will have all 1s. For example, the image below corresponds to the connectivity matrix on the right.




```
C =

     1     1     0     0     1     1
     1     1     1     1     0     0
     0     1     1     0     0     0
     0     1     0     1     0     0
     1     0     0     0     1     0
     1     0     0     0     0     1
```

2.2 Drawing a Tree

The next step is to repeat this design using *recursion*. This is where the "recursive connection points" play a role. Let's call the first branch the root branch. The last two points of the root, marked by open blue circles (see example diagram above), are points where new branches will be drawn. These new branches are identical in shape and orientation to the original branch. A blue circle (recursive connection point) of the root branch is the origin of a new branch. So the root branch shown above will have two new branches, each aligned such that its red asterisk is at the position of a blue circle of the root branch. Then each new branch will have its own two new branches, and so on. This process continues for a set number of "levels." For example, the diagram on page 1 is generated with five levels; a single branch as shown in the diagram above is level 1. Such a self repeating design is drawn using *recursion*.

Complete the function below to create this structure.

```
function drawTree(a, b, x, y, C, color, level)
% Recursively draw a tree:  draw one branch and then at each defined
% recursion point draw another branch using recursion.
% (a,b) is location of the first point of the branch.
% x is vector of x coordinates of points in the branch.
% y is vector of y coordinates of points in the branch.
% C is symmetric connectivity matrix indicating which points connect.
% color is a vector or character indicating the color of the tree.
% level is the number of times further recursion takes place.
```

This function will draw the designed branch at `(a,b)` and will use recursion to draw further branches to complete the tree.

Note that `drawTree` does not call `drawBranch`. As shown in `drawTreeGraphics`, `drawBranch` is called first for the user to make a design and it returns `x`, `y`, and `C`. Then `drawTree` is called with `x`, `y`, and `C`, which represent the user design, as three of the seven arguments. Run the script `drawTreeGraphics` to test your functions. The last statement in `drawTreeGraphics` saves your tree graphic as a jpeg file called `myTree.jpg`. Design your favorite tree!

For Part 2 of Project 6, submit your files `drawBranch.m`, `drawTree.m`, and your favorite version of `myTree.jpg` on CMS. We will put together a gallery of intriguing designs!