Previous Lecture:
- File I/O, use of cell array

Today’s Lecture:
- Structures
- Structure array (i.e., an array of structures)
- A structure with array fields

Announcements:
- Project 5 due Thurs 11/6 at 11pm. Reduced late penalty of 5% applies to submission made up to 11/7 at 11pm
- Prelim 2 on Thurs 11/13 at 7:30pm. Email Randy Hess (rbh27) now if you have an exam conflict (include the course and instructor info of the conflicting exam)

Data are often related
- A point in the plane has an x coordinate and a y coordinate.
- If a program manipulates lots of points, there will be lots of x's and y's.
- Anticipate clutter. Is there a way to “package” the two coordinate values?

A point in the plane has an x coordinate and a y coordinate.
If a program manipulates lots of points, there will be lots of x’s and y’s.
Anticipate clutter. Is there a way to “package” the two coordinate values?

Packaging affects thinking

Our Reasoning Level:
P and Q are points. Compute the midpoint M of the connecting line segment.

Behind the scenes we do this:
\[ M_x = \frac{(P_x + Q_x)}{2} \]
\[ M_y = \frac{(P_y + Q_y)}{2} \]

We’ve seen this before: functions are used to “package” calculations.

This packaging (a type of abstraction) elevates the level of our reasoning and is critical for problem solving.

Example: a Point structure

% p1 is a Point
p1.x = 3;
p1.y = 4;

% p2 is another Point
p2.x = -1;
p2.y = 7;

A Point has two properties—fields—x and y

Working with Point structures

p1.x=3; p1.y=4;
p2.x=-1; p2.y=7;

% Distance between points p1 and p2
D = sqrt((p1.x-p2.x)^2 + (p1.y-p2.y)^2);

Note that p1.x, p1.y, p2.x, p2.y participate in the calculation as variables—because they are.

Different ways to create a structure

% Create a struct by assigning field values
p1.x = 3;
p1.y = 4;

% Create a struct with built-in function
p2 = struct('x',-1, 'y',7);

p2 is a structure.
The structure has two fields.
Their names are x and y.
They are assigned the values -1 and 7.
Assigning to a field in a structure

\[ p1.x = p1.y^2; \]

Assigns the value 16 to \( p1.x \)

A structure can have fields of different types

\[ A = \text{struct}('\text{sname}', 'New York',... 'capital', 'Albany',... 'pop', 15.5) \]

- Can have combinations of string fields and numeric fields
- Arguments are given in pairs: a field name, followed by the value

Legal/Illegal maneuvers

\[ Q = \text{struct}'x',5,'y',6) \]
\[ R = Q \quad \% \text{Legal. } R \text{ is a copy of } Q \]
\[ S = (Q+R)/2 \quad \% \text{Illegal. Must access the fields to do calculations} \]
\[ P = \text{struct}'x',3,'y') \quad \% \text{Illegal. Args must be in pairs (field name followed by field value)} \]
\[ P = \text{struct}'x',3,'y',[]' \quad \% \text{Legal. Use } [] \text{ as place holder} \]

Structures in functions

\[
\text{function } d = \text{dist}(P,Q) \\
\% P and Q are points (structure). \% d is the distance between them. \\
\]\
\[ d = \sqrt{(P.x-Q.x)^2 + ... (P.y-Q.y)^2}; \]

Example “Make” Function

\[
\text{function } P = \text{MakePoint}(x,y) \\
\% P is a point with P.x and P.y assigned the values x and y. \\
\text{P} = \text{struct}'x',x,'y',y'; \]

Then in a script or some other function...

\[
a=10; \ b=\text{rand}; \ P=\text{MakePoint}(a,b); \% \text{create a point structure according to definition in MakePoint function} \]
Another function that has structure parameters

```matlab
function DrawLine(P,Q,c)
    % P and Q are points (structure).
    % Draws a line segment connecting
    % P and Q. Color is specified by c.
    plot([P.x Q.x],[P.y Q.y],c)
end
```

Pick Up Sticks

```matlab
s = 'rgbmcy';
for k=1:100
    P = MakePoint(randn,randn);
    Q = MakePoint(randn,randn);
    c = s(ceil(6*rand));
    DrawLine(P,Q,c)
end
```

Generates two random points and connect them using one of six colors chosen randomly.

Structure Arrays

- An array whose components are structures
- All the structures must be the same (have the same fields) in the array
- Example: an array of points (point structures)

```matlab
P = [MakePoint(.50,.86), MakePoint(1.5,.91), MakePoint(.4,.28), MakePoint(2.5,1.8)];
P(1) = MakePoint(.50,.86)
```

Example: all possible triangles

- Place n points uniformly around the unit circle.
- Draw all possible unique triangles obtained by connecting these points 3-at-a-time.

```matlab
function P = CirclePoints(n)
    % P is array of n point structs; the
    % points are evenly spaced on unit circle
    theta = 2*pi/n;
    for k=1:n
        c = cos(theta*k);
        s = sin(theta*k);
        P(k) = MakePoint(c,s);
    end
end
```
function DrawTriangle(U,V,W,c)
% Draw c-colored triangle;
% triangle vertices are points U,
% V, and W.
fill([U.x V.x W.x], ...
     [U.y V.y W.y], c)

(i,j,k) = (1,3,6)
1 2 3
1 2 4
1 2 5
1 2 6
1 3 4
1 3 5
1 3 6
1 4 5
1 4 6
1 5 6
2 3 4
2 3 5
2 3 6
2 4 5
2 4 6
2 5 6
3 4 5
3 4 6
3 5 6
4 5 6

All possible (i,j,k) combinations but avoid duplicates.
Loop index values have this relationship \( i < j < k \)

% Given \( P \), an array of point structures
for i=1:n
    for j=i+1:n
        for k=j+1:n
            DrawTriangle(P(i),P(j),P(k),'m')
            pause
            DrawTriangle(P(i),P(j),P(k),'k')
        end
    end
end

Bad! \( i, j, \) and \( k \) should be different, and
there should be no duplicates

% Drawing on a black background
for i=1:n-2
    for j=i+1:n-1
        for k=j+1:n
            DrawTriangle(P(i),P(j),P(k),'m')
            DrawPoints(P)
            pause
            DrawTriangle(P(i),P(j),P(k),'k')
        end
    end
end

Both versions print all possible, unique combinations of
(i,j,k), but the left fragment is far more efficient

All possible unique triangles

% Drawing on a black background
for i=1:n-2
    for j=i+1:n-1
        for k=j+1:n
            if i<j && j<k
                DrawTriangle(P(i),P(j),P(k),'m')
                DrawPoints(P)
                pause
                DrawTriangle(P(i),P(j),P(k),'k')
            end
        end
    end
end

The following triangles are the same: (1,3,6), (1,6,3),
(3,1,6), (3,6,1), (6,1,3), (6,3,1)
Still get the same result if all three loop indices end with \( n \)?

A: Yes  B: No

```matlab
for i=1:n
    for j=i+1:n
        for k=j+1:n
            disp([i j k])
        end
    end
end
```

Structures with array fields

Let’s develop a structure that can be used to represent a colored disk. It has four fields:

- \( xc \): x-coordinate of center
- \( yc \): y-coordinate of center
- \( r \): radius
- \( c \): rgb color vector

Examples:

- \( D1 = \text{struct}('xc',1,'yc',2,'r',3,...'
- \( 'c',[1 0 1])\)
- \( D2 = \text{struct}('xc',4,'yc',0,'r',1,...'
- \( 'c',[.2 .5 .3])\)

Example: Averaging two disks

\( D1 \) and \( D2 \) are disk structures.

\% Average is:

\[
\begin{align*}
    r &= (D1.r + D2.r) / 2; \\
    xc &= (D1.xc + D2.xc)/2; \\
    yc &= (D1.yc + D2.yc)/2; \\
    c &= (D1.c + D2.c) / 2;
\end{align*}
\]

\% The average is also a disk

\( D = \text{struct}('xc',xc,'yc',yc,'r',r,'c',c)\)

Example: How do you assign to \( g \) the green-color component of disk \( D \)?

\( D = \text{struct}('xc',3.5,'yc',2,...'
- \( 'r',1.0,'c',[.4 .1 .5])\)

A: \( g = D.g; \)
B: \( g = D.c.g; \)
C: \( g = D.c.2; \)
D: \( g = D.c(2); \)
E: other

Different kinds of abstraction

- Packaging procedures (program instructions) into a function
  - A program is a set of functions executed in the specified order
  - Data is passed to (and from) each function
- Packaging data into a structure
  - Elevates thinking
  - Reduces the number of variables being passed to and from functions
- Packaging data, and the instructions that work on those data, into an object
  - A program is the interaction among objects
  - Object-oriented programming (OOP) focuses on the design of data-instructions groupings