• Previous lecture:
  – Structure & structure array
• Today’s lecture:
  – More on structs
  – Introduction to objects and classes
• Announcements:
  – Project 5 due tonight at 11pm
  – Do Exercise 11 question 3.1 and 3.2. Submit them on paper at
    the beginning of your next discussion
  – Prelim 2 on Thurs, Nov 10 at 7:30pm
  – Prelim 2 topics: end with Project 5 and Lecture 19, i.e., will
    NOT include structs and OOP
  – Review: Re-do discussion/lecture examples, don’t just read
    them! Study using posted review Qs. Test yourself using posted
    old exams.
  – Optional review sessions: Sun 1-2:30pm and Wedn 8-9:30pm;
    see website for details

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

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

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 4 \\
1 & 2 & 5 \\
1 & 2 & 6 \\
1 & 3 & 4 \\
1 & 3 & 5 \\
1 & 3 & 6 \\
1 & 4 & 5 \\
1 & 4 & 6 \\
1 & 5 & 6 \\
2 & 3 & 4 \\
2 & 3 & 5 \\
2 & 3 & 6 \\
2 & 4 & 5 \\
2 & 4 & 6 \\
2 & 5 & 6 \\
3 & 4 & 5 \\
3 & 4 & 6 \\
3 & 5 & 6 \\
4 & 5 & 6 \\
\end{array}
\]

Still get the same result if all three loop indices end
with \( n \)?

A: Yes  B: No

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 4 \\
1 & 2 & 5 \\
1 & 2 & 6 \\
1 & 3 & 4 \\
1 & 3 & 5 \\
1 & 3 & 6 \\
1 & 4 & 5 \\
1 & 4 & 6 \\
1 & 5 & 6 \\
2 & 3 & 4 \\
2 & 3 & 5 \\
2 & 3 & 6 \\
2 & 4 & 5 \\
2 & 4 & 6 \\
2 & 5 & 6 \\
3 & 4 & 5 \\
3 & 4 & 6 \\
3 & 5 & 6 \\
4 & 5 & 6 \\
\end{array}
\]

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

\[
\begin{align*}
\text{xc:} & \text{ x-coordinate of center} \\
\text{yc:} & \text{ y-coordinate of center} \\
r: & \text{ radius} \\
c: & \text{ rgb color vector}
\end{align*}
\]

Examples:

\[
\begin{align*}
D1 = \text{struct}(`\text{xc}',1,`\text{yc}',2,`r',3,\ldots
&`c',[1 0 1]); \\
D2 = \text{struct}(`\text{xc}',4,`\text{yc}',0,`r',1,\ldots
&`c',[.2 .5 .3]);
\end{align*}
\]

Example: Averaging two disks

\[
\begin{array}{c}
\text{D1} \\
\text{D2} \\
\text{D} \\
\end{array}
\]

Still get the same result if all three loop indices end
with \( n \)?

A: Yes  B: No

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 4 \\
1 & 2 & 5 \\
1 & 2 & 6 \\
1 & 3 & 4 \\
1 & 3 & 5 \\
1 & 3 & 6 \\
1 & 4 & 5 \\
1 & 4 & 6 \\
1 & 5 & 6 \\
2 & 3 & 4 \\
2 & 3 & 5 \\
2 & 3 & 6 \\
2 & 4 & 5 \\
2 & 4 & 6 \\
2 & 5 & 6 \\
3 & 4 & 5 \\
3 & 4 & 6 \\
3 & 5 & 6 \\
4 & 5 & 6 \\
\end{array}
\]
Example: compute “average” of two disks
% D1 and D2 are disk structures.
% Average is:
r  = (D1.r  + D2.r) /2;
x = (D1.xc + D2.xc)/2;
y = (D1.yc + D2.yc)/2;
c  = (D1.c  + D2.c) /2;
% The average is also a disk
D = struct('xc',xc,'yc',yc,'r',r,'c',c)

A structure’s field can hold a structure
A = MakePoint(2,3)
B = MakePoint(4,5)
L = struct('P',A,'Q',B)
- This could be used to represent a line segment
  with endpoints P and Q, for instance
- Given the MakePoint function to create a point
  structure, what is x below?
  x = L.P.y;

A card game, developed in two ways
- Develop the algorithm—the logic—of the card game:
  - Set up a deck as an array of cards. (First, choose
    representation of cards.)
  - Shuffle the cards
  - Deal cards to players
  - Evaluate each player’s hand to determine
    winner

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

A card game, developed in two ways
- Identify “objects” in the game and define each:
  - Card
    - Properties: suit, rank
    - Actions: compare, show
  - Deck
    - Property: array of Cards
    - Actions: shuffle, deal, get #cards left
  - Hand
    - Actions: ...
  - Player
    - Actions: ...
- Then write the game—the algorithm—using objects of
  the above “classes”

How do you assign to g the green-color component of disk D?
D= 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
Notice the two steps involved in OOP?

- Define the classes (of the objects)
  - Identify the properties (data) and actions (methods, i.e., functions) of each class

- Create the objects (from the classes) that are then used—that interact with one another

Defining a class ≠ creating an object

- A class is a specification
  - E.g., a cookie cutter specifies the shape of a cookie

- An object is a concrete instance of the class
  - Need to apply the cookie cutter to get a cookie (an instance, the object)
  - Many instances (cookies) can be made using the class (cookie cutter)
  - Instances do not interfere with one another. E.g., biting the head off one cookie doesn't remove the heads of the other cookies

Example class: Rectangle

- Properties:
  - xLL, yLL, width, height

- Methods (actions):
  - Calculate area
  - Calculate perimeter
  - Draw
  - Intersect (the intersection between two rectangles is a rectangle!)

Example class: Time

- Properties:
  - Hour, minute, second

- Methods (actions):
  - Show (e.g., display in hh:mm:ss format)
  - Advance (e.g., advance current time by some amount)

Example class: Window (e.g., dialog box)

- Properties:
  - Size, title, option buttons, input dialog …

- Methods (actions):
  - Show
  - Resize
  - …

Matlab supports procedural and object-oriented programming

- We have been writing procedural programs—focusing on the algorithm, implemented as a set of functions

- We have used objects in Matlab as well, e.g., graphics
  - A plot is a “handle graphics” object
    - Can produce plots without knowing about objects
    - Knowing about objects gives more possibilities

Many such useful classes have been predefined!

See demoPlotObj.m
The `plot` handle graphics object in Matlab

\[
x = ...; \quad y = ...;
\]
\[plot(x, y)\] creates a graphics object

- In the past we focused on the visual produced by that command. If we want the visual to look different we make another plot.
- We can actually hang on to the graphics object—store its “handle”—so that we can later make changes to that object.

Objects of the same class have the same properties

\[
x = 1:10;
\%
Two separate graphics objects:
plot(x, \sin(x), 'k-')
plot(x(1:5), 2.^x, 'm-*')
\]

- Both objects have some x-data, some y-data, some line style, and some marker style. These are the properties of one kind, or class, of the objects (plots)
- The values of the properties are different for the individual objects

See demoPlotObj.m

Object-Oriented Programming

- First design and define the classes (of the objects)
  - Identify the properties (data) and actions (methods, i.e., functions) of each class
- Then create the objects (from the classes) that are then used, that interact with one another