- Previous Lecture:
  - 2-d array examples

- Today’s Lecture:
  - Working with images

- Announcements:
  - Discussion this week in the UP B7 computer lab
  - Prelim 1 to be returned at end of lecture.
    Unclaimed papers (and those on which student didn’t indicate the lecture time) can be picked up during consulting hours (Su-R 5-10p) at ACCEL Green Rm (Carpenter Hall) starting at 5pm today
Who Can Fill the Order?

<table>
<thead>
<tr>
<th>Inv</th>
<th>38</th>
<th>5</th>
<th>99</th>
<th>34</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
</tr>
<tr>
<td>PO</td>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
</tr>
</tbody>
</table>

- Yes
- No
- Yes
Wanted: A True/False Function

DO is “true” if factory $i$ can fill the order. DO is “false” if factory $i$ cannot fill the order.
Example: Check inventory of factory 2

<table>
<thead>
<tr>
<th>Inv</th>
<th>38</th>
<th>5</th>
<th>99</th>
<th>34</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
</tr>
<tr>
<td>PO</td>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
</tr>
</tbody>
</table>

Method 1: check the inventory for every product
### Initialization

<table>
<thead>
<tr>
<th>Inv</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>5</td>
<td>99</td>
<td>34</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PO</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Still True…

\[ DO = DO \land ( Inv(2,1) \geq PO(1) ) \]
Still True…

\[
\text{DO} = \text{DO} \land (\text{Inv}(2,2) \geq \text{PO}(2))
\]

<table>
<thead>
<tr>
<th></th>
<th>Inv</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>5</td>
<td>99</td>
<td>34</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PO</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Still True...

\[
\text{DO} = \text{DO} && (\text{Inv}(2,3) \geq \text{PO}(3))
\]
No Longer True...

\[ \text{DO} = \text{DO} \land (\text{Inv}(2,4) \geq \text{PO}(4)) \]
Stay False…

\[
\text{DO} = \text{DO} \land (\text{Inv}(2,5) \geq \text{PO}(5))
\]
function  DO = iCanDo(i,Inv,PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false

nProd = length(PO);
DO = 1;
for j = 1:nProd
    DO = DO && ( Inv(i,j) >= PO(j) );
end
Encapsulate...

function DO = iCanDo(i, Inv, PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false
nProd = length(PO);
j = 1;
while j<=nProd && Inv(i, j)>=PO(j)
    j = j+1;
end
DO = _______;

function DO = iCanDo(i, Inv, PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false
nProd = length(PO);
j = 1;
while j<=nProd && Inv(i, j)>=PO(j) 
    j = j+1;
end
DO = __________;

DO should be true when...
A j < nProd
B j == nProd
C j > nProd
function DO = iCanDo(i, Inv, PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false
nProd = length(PO);
j = 1;
while j<=nProd && Inv(i,j)>=PO(j)
    j = j+1;
end
DO = (j>nProd);
Back To Finding the Cheapest

\[
iBest = 0; \quad \text{minBill} = \text{inf};
\]

for \( i=1:n\text{Fact} \)

\[
i\text{Bill} = \text{iCost}(i,C,\text{PO});
\]

if \( i\text{Bill} < \text{minBill} \)

\% Found an Improvement

\[
i\text{Best} = i; \quad \text{minBill} = i\text{Bill};
\]

end

end

Don’t bother with this unless there is sufficient inventory.
Back To Finding the Cheapest

\[ i_{\text{Best}} = 0; \ \text{minBill} = \infty; \]
\[ \text{for } i=1:n_{\text{Fact}} \]
\[ \quad \text{if } i_{\text{CanDo}}(i, \text{Inv}, \text{PO}) \]
\[ \quad \quad i_{\text{Bill}} = i_{\text{Cost}}(i, \text{C}, \text{PO}); \]
\[ \quad \quad \text{if } i_{\text{Bill}} < \text{minBill} \]
\[ \quad \quad \quad \% \ \text{Found an Improvement} \]
\[ \quad \quad \quad i_{\text{Best}} = i; \ \text{minBill} = i_{\text{Bill}}; \]
\[ \quad \text{end} \]
\[ \text{end} \]

See Cheapest.m for alternative implementation
A picture as a matrix

1458-by-2084

<table>
<thead>
<tr>
<th>150</th>
<th>149</th>
<th>152</th>
<th>153</th>
<th>152</th>
<th>155</th>
</tr>
</thead>
<tbody>
<tr>
<td>151</td>
<td>150</td>
<td>153</td>
<td>154</td>
<td>153</td>
<td>156</td>
</tr>
<tr>
<td>153</td>
<td>151</td>
<td>155</td>
<td>156</td>
<td>155</td>
<td>158</td>
</tr>
<tr>
<td>154</td>
<td>153</td>
<td>156</td>
<td>157</td>
<td>156</td>
<td>159</td>
</tr>
<tr>
<td>156</td>
<td>154</td>
<td>158</td>
<td>159</td>
<td>158</td>
<td>161</td>
</tr>
<tr>
<td>157</td>
<td>156</td>
<td>159</td>
<td>160</td>
<td>159</td>
<td>162</td>
</tr>
</tbody>
</table>
Images can be encoded in different ways

- Common formats include
  - JPEG: Joint Photographic Experts Group
  - GIF: Graphics Interchange Format
- Data are compressed
- We will work with jpeg files:
  - `imread`: read a .jpg file and convert it to a “normal numeric” array that we can work with
  - `imwrite`: write an array into a .jpg file (compressed data)
Grayness: a value in [0..255]

0 = black
255 = white

These are integer values
Type: uint8
Let’s put a picture in a frame

Things to do:

1. Read `bwduck.jpg` from memory and convert it into an array
2. Show the original picture
3. Assign a gray value (frame color) to the “edge pixels”
4. Show the manipulated picture
Reading a jpeg file and displaying the image

% Read jpg image and convert to
% an array P

\[
P = \texttt{imread('bwduck.jpg')};
\]

% Show the data in array P as
% an image

\[
\texttt{imsh ow}(P)
\]
% Frame a grayscale picture

P = imread('bwduck.jpg');
imshow(P)

% Change the "frame" color

imshow(P)
% Frame a grayscale picture

P = imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width = 50;
frameColor = 200; % light gray

imshow(P)
% Frame a grayscale picture

P = imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width = 50;
frameColor = 200; % light gray
[nr,nc] = size(P);
for r = 1:nr
    for c = 1:nc
        % At pixel (r,c)
        
    end
end
end
imshow(P)
% Frame a grayscale picture

P = imread('bwduck.jpg');
iimshow(P)

% Change the "frame" color
width = 50;
frameColor = 200; % light gray
[nr, nc] = size(P);
for r = 1:nr
    for c = 1:nc
        % At pixel (r,c)
        if  r<=width || r>nr-width || ... 
            c<=width || c>nc-width
            P(r,c) = frameColor;
        end
    end
end
iimshow(P)

Things to consider...
1. What is the type of the values in P?
2. Can we be more efficient?
Accessing a submatrix

- $M$ refers to the whole matrix
- $M(3,5)$ refers to one component of $M$
Accessing a submatrix

- \( M \) refers to the whole matrix
- \( M(3,5) \) refers to one component of \( M \)
- \( M(2:3,3:5) \) refers to a submatrix of \( M \)

\[
\begin{array}{cccccc}
2 & -1 & .5 & 0 & -3 \\
3 & 8 & 6 & 7 & 7 \\
5 & -3 & 8.5 & 9 & 10 \\
52 & 81 & .5 & 7 & 2 \\
\end{array}
\]
Grayness: a value in $[0..255]$  

0 = black  
255 = white  

These are integer values  
Type: uint8
A color picture is made up of RGB matrices $\rightarrow$ 3-d array

E.g., color image data is stored in a 3-d array $A$:

\[
0 \leq A(i,j,1) \leq 255 \\
0 \leq A(i,j,2) \leq 255 \\
0 \leq A(i,j,3) \leq 255
\]
A color picture is made up of RGB matrices $\rightarrow$ 3-d array

Operations on images amount to operations on matrices!
Example: Mirror Image

1. Read LawSchool.jpg from memory and convert it into an array.
2. Manipulate the Array.
3. Convert the array to a jpg file and write it to memory.
Reading and writing jpg files

% Read jpg image and convert to
% a 3D array A
A = imread('LawSchool.jpg');

% Write 3D array B to memory as
% a jpg image
imwrite(B,'LawSchoolMirror.jpg')
A 3-d array as 3 matrices

\[ [nr, nc, np] = \text{size}(A) \]  \% dimensions of 3-d array \( A \)

#rows  #columns  #layers (pages)

\[ A(1:nr,1:nc,1) \]

4-by-6  \( M1 = A(:, :, 1) \)

4-by-6  \( M2 = A(:, :, 2) \)

4-by-6  \( M3 = A(:, :, 3) \)
%Store mirror image of A in array B

\[ [nr, nc, np] = \text{size}(A); \]
\[ \text{for } r = 1:nr \]
\[ \quad \text{for } c = 1:nc \]
\[ \quad \quad B(r, c) = A(r, nc-c+1); \]
\[ \quad \end \]
\[ \end \]
%Store mirror image of A in array B

[nr,nc,np]= size(A);
for r= 1:nr
    for c= 1:nc
        for p= 1:np
            B(r,c,p)= A(r,nc-c+1,p);
        end
    end
end
% Make mirror image of A -- the whole thing

A = imread('LawSchool.jpg');
[nr, nc, np] = size(A);

for r = 1:nr
    for c = 1:nc
        for p = 1:np
            B(r, c, p) = A(r, nc-c+1, p);
        end
    end
end

imshow(B) % Show 3-d array data as an image
imwrite(B,'LawSchoolMirror.jpg')
% Make mirror image of A -- the whole thing

A = imread('LawSchool.jpg');
[nr,nc,np] = size(A);

B = zeros(nr,nc,np);
B = uint8(B); % Type for image color values

for r = 1:nr
    for c = 1:nc
        for p = 1:np
            B(r,c,p) = A(r,nc-c+1,p);
        end
    end
end

imshow(B)  % Show 3-d array data as an image
imwrite(B,'LawSchoolMirror.jpg')
Vectorized code simplifies things…
Work with a whole column at a time
Vectorized code simplifies things…
Work with a whole column at a time
Vectorized code simplifies things…
Work with a whole column at a time
Vectorized code simplifies things…
Work with a whole column at a time
Vectorized code simplifies things…
Work with a whole column at a time
Vectorized code simplifies things…
Work with a whole column at a time

Column c in B
is column nc-c+1 in A
Consider a single matrix (just one layer)

\[
[nr, nc, np] = \text{size}(A);
\]

for \( c = 1 : nc \)

\[
B(\text{all rows}, c) = A(\text{all rows}, nc+1-c);
\]

end
Consider a single matrix (just one layer)

\[
[nr, nc, np] = \text{size}(A); \\
\text{for } c = 1:nc \\
\hspace{1cm} B(1:nr, c) = A(1:nr, nc+1-c); \\
\text{end}
\]
Consider a single matrix (just one layer)

```
[nr,nc,np] = size(A);
for c= 1:nc
    B( : ,c ) = A( : ,nc+1-c );
end
```

The colon says “all indices in this dimension.” In this case it says “all rows.”
Now repeat for all layers

\[
[nr, nc, np] = \text{size}(A);
\]

\[
\text{for } c = 1 : nc
\]

\[
B( :, c, 1) = A( :, nc+1-c, 1)
\]

\[
B( :, c, 2) = A( :, nc+1-c, 2)
\]

\[
B( :, c, 3) = A( :, nc+1-c, 3)
\]

\[
\text{end}
\]
Vectorized code to create a mirror image

\[
A = \text{imread('LawSchool.jpg')} \\
\text{[nr,nc,np] = size(A);} \\
\text{for c = 1:nc} \\
\quad B(:,c,1) = A(:,nc+1-c,1) \\
\quad B(:,c,2) = A(:,nc+1-c,2) \\
\quad B(:,c,3) = A(:,nc+1-c,3) \\
\text{end} \\
imwrite(B,'LawSchoolMirror.jpg')
\]
Prelim 1

- Q1: Program trace (vectors) & function scope
- Q2: random numbers, for-loop pattern (for vector)
- Q3: accumulation pattern (similar to P2 pi sequence)
- Q4: simulation involving while-loop, rand int, if-construct; function and function call; building a vector of non-determined length (similar to random walk and P3 simulation)
- Q5: nested loops, drawing a 2-d pattern (similar to P2 café wall)

if score>80
    celebrate, look up solutions and learn from mistakes
elseif score>60
    *re-do the open questions* that you got wrong first; then read solutions
else
    *see course staff one-on-one* to re-do the questions; *avoid the solutions!*
end

If your paper isn’t here, pick it up from Matlab consultants in ACCEL Green Rm during consulting hrs (starting today at 5pm)