Previous Lecture:
- 2-d array examples
- char and string

Today's Lecture:
- More on char and string
- Image processing

Announcements:
- Discussion this week in computer lab
- Pick up Prelim 1 during consulting hours (Su-Th 5-10p) at ACCEL Green Rm (Carpenter Hall). Regrade request must be submitted to a consultant by Thursday 10pm at consulting.

---

ASCII characters
(American Standard Code for Information Interchange)

<table>
<thead>
<tr>
<th>ascii code</th>
<th>Character</th>
<th>ascii code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>65</td>
<td>‘A’</td>
<td>48</td>
<td>‘0’</td>
</tr>
<tr>
<td>66</td>
<td>‘B’</td>
<td>49</td>
<td>‘1’</td>
</tr>
<tr>
<td>67</td>
<td>‘C’</td>
<td>50</td>
<td>‘2’</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>90</td>
<td>‘Z’</td>
<td>57</td>
<td>‘9’</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

---

The ASCII Table

<table>
<thead>
<tr>
<th>Char</th>
<th>Ascii</th>
<th>Char</th>
<th>Ascii</th>
<th>Char</th>
<th>Ascii</th>
<th>Char</th>
<th>Ascii</th>
<th>Char</th>
<th>Ascii</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00000</td>
<td>a</td>
<td>00101</td>
<td>A</td>
<td>00000</td>
<td>a</td>
<td>00101</td>
<td>A</td>
<td>00000</td>
</tr>
<tr>
<td>B</td>
<td>00001</td>
<td>b</td>
<td>00102</td>
<td>B</td>
<td>00001</td>
<td>b</td>
<td>00102</td>
<td>B</td>
<td>00001</td>
</tr>
<tr>
<td>C</td>
<td>00010</td>
<td>c</td>
<td>00103</td>
<td>C</td>
<td>00010</td>
<td>c</td>
<td>00103</td>
<td>C</td>
<td>00010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>00011</td>
<td>d</td>
<td>00104</td>
<td>D</td>
<td>00011</td>
<td>d</td>
<td>00104</td>
<td>D</td>
<td>00011</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>00012</td>
<td>e</td>
<td>00105</td>
<td>E</td>
<td>00012</td>
<td>e</td>
<td>00105</td>
<td>E</td>
<td>00012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>00013</td>
<td>f</td>
<td>00106</td>
<td>F</td>
<td>00013</td>
<td>f</td>
<td>00106</td>
<td>F</td>
<td>00013</td>
</tr>
</tbody>
</table>

---

Character vs ASCII code

```matlab
str = 'Age 19'
% a 1-d array of characters
code = double(str)
% convert chars to ascii values
str1 = char(code)
% convert ascii values to chars
```

---

Arithmetic and relational ops on characters

- ‘c’ - ‘a’ gives 2
- ‘6’ - ‘5’ gives 1
- letter1 = 'e'; letter2 = 'f';
- letter1 - letter2 gives -1
- ‘c’ > ‘a’ gives true
- letter1 == letter2 gives false
- ‘A’ + 2 gives 67
- char('A'+2) gives ‘C’

---

Example: toUpper

Write a function `toUpper(cha)` to convert character `cha` to upper case if `cha` is a lower case letter. Return the converted letter. If `cha` is not a lower case letter, simply return the character `cha`.

**Hint:** Think about the distance between a letter and the base letter ‘a’ (or ‘A’). E.g.,
```
 a b c d e f g h ...

distance = 'g'-'a' = 6 = 'G'-'A'
```

```
A B C D E F G H ...
```

Of course, do not use Matlab function `upper`!
function up = toUpper(cha)  
% up is the upper case of character cha.  
% If cha is not a letter then up is just cha.  
up = cha;  
cha is lower case if it is between 'a' and 'z'

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 = imread('bwduck.jpg');

% Show the data in array P as  
% an image
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
    if r <= width || r > nr - width || ... 
      c <= width || c > nc - width 
      P(r, c) = frameColor;
    end
  end
end
imshow(P)

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

A color picture is made up of RGB matrices \rightarrow 3-d array

<table>
<thead>
<tr>
<th>Color image</th>
<th>3-d Array</th>
</tr>
</thead>
</table>
| ![Color Image](image1.png) | \(0 \leq A(i,j,1) \leq 255\)  
| ![Color Image](image2.png) | \(0 \leq A(i,j,2) \leq 255\)  
| ![Color Image](image3.png) | \(0 \leq A(i,j,3) \leq 255\)  

Operations on images amount to operations on matrices!

Example: color \rightarrow black and white

Can “average” the three color values to get one gray value.

Averaging the RGB values to get a gray value
\[
\text{R} \rightarrow 0.3R + 0.59G + 0.11B
\]
\[
\text{G} \rightarrow 0.3R + 0.59G + 0.11B
\]
\[
\text{B} \rightarrow 0.3R + 0.59G + 0.11B
\]

for i = 1:m
  for j = 1:n
    M(i,j) = 0.3*R(i,j) + 0.59*G(i,j) + 0.11*B(i,j)
  end
end

scalar operation
Averaging the RGB values to get a gray value

\[
M(i,j) = 0.3A(i,j,1) + 0.59A(i,j,2) + 0.11A(i,j,3)
\]

Here are 2 ways to calculate the average. Are gray value matrices \(g\) and \(h\) the same given image data \(A\)?

\[
\begin{align*}
g(r,c) &= \frac{A(r,c,1)}{3} + \frac{A(r,c,2)}{3} + \frac{A(r,c,3)}{3}; \\
h(r,c) &= \frac{A(r,c,1) + A(r,c,2) + A(r,c,3)}{3};
\end{align*}
\]

A: yes  B: no

Matlab has a built-in function to convert from color to grayscale, resulting in a 2-d array:

\[
B = \text{rgb2gray}(A)
\]

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 \(A\)
\[
A = \text{imread}('LawSchool.jpg');
\]

% Write 3D array \(B\) to memory as a jpg image
\[
\text{imwrite}(B,'LawSchoolMirror.jpg')
\]

A 3-d array as 3 matrices

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

\[
\begin{align*}
M1 &= A(:,:,1) \\
M2 &= A(:,:,2) \\
M3 &= A(:,:,3)
\end{align*}
\]
%Store mirror image of A in array B

```
[r, n, p] = size(A);
for r = 1:r
    for c = 1:c
        B(r, c) = A(r, n-c+1);
    end
end
```

Both fragments create a mirror image of A.

true
false

% Make mirror image of A -- the whole thing

```
A = imread('LawSchool.jpg');
[r, n, p] = size(A);

B = zeros(r, n, p);
B = uint8(B); % Type for image color values
for r = 1:r
    for c = 1:c
        for p = 1:p
            B(r, c, p) = A(r, n-c+1, p);
        end
    end
end
imshow(B)  % Show 3-d array data as an image
imwrite(B, 'LawSchoolMirror.jpg')
```