- **Previous Lecture:**
  - Image processing
    - 3-d array, computing with type `uint8`, vectorized code
  - Read 12.4 of textbook (image processing, type `uint8`)

- **Today’s Lecture:**
  - Computing with characters (arrays of type `char`)
  - Review top-down design for program development
  - Linear search

- **Announcements:**
  - Project 4A due tonight; Project 4B due Tue, Apr 13
  - Review Prelim 1; regrade requests due Mon
Deal with boundary issues – moving window

% Get C, the radius r
% neighborhood of pixel (i,j)

\[
i_{\text{Min}} = \max(1, i-r)
\]

\[
i_{\text{Max}} = \min(nr, i+r)
\]

\[
j_{\text{Min}} = \max(1, j-r)
\]

\[
j_{\text{Max}} = \min(nc, j+r)
\]

\[
C = A(i_{\text{Min}}:i_{\text{Max}}, j_{\text{Min}}:j_{\text{Max}})
\]

See Insight §12.4 for complete code: MedianFilter.m
\[ B = \text{medianFilter}(A, 3) \]
Mean Filter with radius 3
Mean Filter with radius 10
Mean filter fails because the mean does not capture representative values.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>149</td>
<td>152</td>
<td>153</td>
<td>152</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>150</td>
<td>153</td>
<td>154</td>
<td>153</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>2</td>
<td>3</td>
<td>156</td>
<td>155</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>2</td>
<td>1</td>
<td>157</td>
<td>156</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>154</td>
<td>158</td>
<td>159</td>
<td>158</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>156</td>
<td>159</td>
<td>160</td>
<td>159</td>
<td>162</td>
<td></td>
</tr>
</tbody>
</table>

- **Mean-filtered values with radius 1 neighborhood:**
  - Mean: 150.33
  - Median: 153.5

- **Median-filtered values with radius 1 neighborhood:**
  - Median: 154
Finding Edges: read example in Sec 12.4

Identify “sharp changes” in image data—a kind of outliers.

Subtracting \texttt{uint8} values correctly to prevent “underflow”

“Thresholding”—use a parameter to control the amount of details extracted from image
Text in programming

• We’ve seen text already
  • `fprintf('Hello world\n'), title('Click here'),` etc.
  • Time to dive into the details

Vocabulary:

• A single letter (or digit, or symbol, or space) is a “character”
• A sequence of characters is called a “string”
  • Could be a word, a sentence, gibberish
Text—sequences of characters often called strings—are important in computation

Numerical data is often encoded in strings. E.g., a file containing Ithaca weather data begins with the string

$$W07629N4226$$

meaning

- Longitude: $76^\circ 29'$ West
- Latitude: $42^\circ 26'$ North

We may need to grab hold of the substring $W07629$, convert $076$ and $29$ to the numeric values $76$ and $29$, and do some computation
Character array (an array of type char)

• We have used strings of characters in programs already:
  • c = input('Give me a letter: ', 's')
  • msg = sprintf('Answer is %d', ans);

• A string is made up of individual characters, so a string is a 1-d array of characters.

• 'CS1112 rocks!' is a character array of length 13; it has 7 letters, 4 digits, 1 space, and 1 symbol.

  \[
  \begin{array}{cccccc}
  C & S & 1 & 1 & 1 & 2 \\
  r & o & c & k & s & !
  \end{array}
  \]

  Row vector of length 13

• Can have 2-d array of characters as well

  \[
  \begin{array}{cccccc}
  C & S & 1 & 1 & 1 & 2 \\
  r & o & c & k & s & !
  \end{array}
  \]

  2×6 matrix
A text sequence is a vector (of characters)

**Vectors**

- **Assignment**
  \[ v= [7, 0, 5]; \]

- **Indexing**
  \[ x= v(3); \]
  \[ v(1)= 1; \]
  \[ w= v(2:3); \]
  \[ x \text{ is 5} \]
  \[ v \text{ is } [1 0 5] \]
  \[ w \text{ is } [0 5] \]

- **: notation**
  \[ v= 2:5; \]
  \[ v \text{ is } [2 3 4 5] \]

- **Appending**
  \[ v= [7 0 5]; \]
  \[ v(4)= 2; \]
  \[ v \text{ is } [7 0 5 2] \]

- **Concatenation**
  \[ v= [v [4 6]]; \]
  \[ v \text{ is } [7 0 5 2 4 6] \]

**Strings**

- **Assignment**
  \[ s= ['h','e','l','l','o']; \]
  \[ s= 'hello'; \]

- **Indexing**
  \[ c= s(2); \]
  \[ s(1)= 'J'; \]
  \[ t= s(2:4); \]
  \[ c \text{ is } 'e' \]
  \[ s \text{ is } 'Jello' \]
  \[ t \text{ is } 'ell' \]

- **: notation**
  \[ s= 'a':'g'; \]
  \[ s \text{ is } 'abcdefg' \]

- **Appending**
  \[ s= 'duck'; \]
  \[ s(5)= 's'; \]
  \[ s \text{ is } 'ducks' \]

- **Concatenation**
  \[ s= [s ' quack']; \]
  \[ s \text{ is } 'ducks quack' \]
Syntax: **Single quotes** enclose char arrays in Matlab

Anything enclosed in single quotes is a string *(even if it looks like something else)*

- `'100'` is a character array (string) of length 3
- `100` is a numeric value
- `'pi'` is a character array of length 2
- `pi` is the built-in constant 3.14159...
- `'x'` is a character (vector of length 1)
- `x` may be a variable name in your program
Types so far: char, double, logical

\begin{itemize}
  \item[] \texttt{a} = 'CS1'
  \item[] \texttt{a} = ['C','S','1']\hfill \texttt{a} is a 1-d array with type \texttt{char} elements. Often called a \textit{string}; NOT the same as a \textit{new} type in Matlab 2017+ called \texttt{string}.
  \item[] \texttt{b} = [3 9] \hfill \texttt{b} is a 1-d array with type \texttt{double} elements. \texttt{double} is the default type for numbers in Matlab. We call \texttt{b} a “numeric array”
  \item[] \texttt{c} = \texttt{uint8(b)} \hfill \texttt{c} is a 1-d array with type \texttt{uint8} elements. We call \texttt{c} a “\texttt{uint8 array}”
  \item[] \texttt{d} = \texttt{rand()} > 0.5 \hfill \texttt{d} is a scalar of the type \texttt{logical}. We call \texttt{d} a “Boolean value”
\end{itemize}
Basic (simple) types in MATLAB

• E.g., char, double, uint8, logical
• Each uses a set amount of memory
  • Each uint8 value uses 8 bits (=1 byte)
  • Each double value uses 64 bits (=8 bytes)
  • Each char value uses 16 bits (=2 bytes)
  • Use function whos to see memory usage by variables in workspace
• Can easily determine amount of memory used by a simple array
  (array of a basic type, where each component stores one simple value)
• Next lecture: Special arrays where each component is a container for a collection of values
Self-check

What is the value of `substr`?

```plaintext
str = 'My hovercraft is full of eels.';
substr = str(19:length(str)-2);
```

A. 'll of eels'
B. 'ull of eel'
C. ['o', 'f', 'e', 'e']
D. [19 20 ... 28]
E. None of the above
Working with gene data → compute on text data

- A gene is a DNA fragment that codes for a protein, e.g.,
  \textbf{ATCGGCTTTTGCACATTCTA}...

- 3-letter DNA “codons” identify the amino acid sequence that defines a protein
A gene is a DNA fragment that codes for a protein, e.g.,

\[
\text{ATCGCTTTTGCACATTCTA...}
\]

- 3-letter DNA “codons” identify the amino acid sequence that defines a protein

- Isoleucine (Ile)
- Alanine (Ala)
- Leucine (Leu)
- Histidine (His)
- Isoleucine (Ile)
- Leucine (Leu)
## The Codon Dictionary

<table>
<thead>
<tr>
<th>Index</th>
<th>Amino Acid</th>
<th>Mnemonic</th>
<th>DNA Codons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alanine</td>
<td>Ala</td>
<td>GCT GCC GCA GCG</td>
</tr>
<tr>
<td>2</td>
<td>Arginine</td>
<td>Arg</td>
<td>CGT CGC CGA CGG AGA AGG</td>
</tr>
<tr>
<td>3</td>
<td>Asparagine</td>
<td>Asn</td>
<td>AAT AAC</td>
</tr>
<tr>
<td>4</td>
<td>Aspartic Acid</td>
<td>Asp</td>
<td>GAT GAC</td>
</tr>
<tr>
<td>5</td>
<td>Cysteine</td>
<td>Cys</td>
<td>TGT TGC</td>
</tr>
<tr>
<td>6</td>
<td>Glutamic Acid</td>
<td>Glu</td>
<td>CAA CAG</td>
</tr>
<tr>
<td>7</td>
<td>Glutamine</td>
<td>Gln</td>
<td>GAA GAG</td>
</tr>
<tr>
<td>8</td>
<td>Glycine</td>
<td>Gly</td>
<td>GGT GGC GGA GGG</td>
</tr>
<tr>
<td>9</td>
<td>Histidine</td>
<td>His</td>
<td>CAT CAC</td>
</tr>
<tr>
<td>10</td>
<td>Isoleucine</td>
<td>Ile</td>
<td>ATT ATC ATA</td>
</tr>
<tr>
<td>11</td>
<td>Leucine</td>
<td>Leu</td>
<td>CTT CTC CTG TTA TTG</td>
</tr>
<tr>
<td>12</td>
<td>Lysine</td>
<td>Lys</td>
<td>AAA AAG</td>
</tr>
<tr>
<td>13</td>
<td>Methionine</td>
<td>Met</td>
<td>ATG</td>
</tr>
<tr>
<td>14</td>
<td>Phenylalanine</td>
<td>Phe</td>
<td>TTT TTC</td>
</tr>
<tr>
<td>15</td>
<td>Proline</td>
<td>Pro</td>
<td>CCT CCC CCA CCG</td>
</tr>
<tr>
<td>16</td>
<td>Serine</td>
<td>Ser</td>
<td>TCT TCC TCA TCG AGT AGC</td>
</tr>
<tr>
<td>17</td>
<td>Threonine</td>
<td>Thr</td>
<td>ACT ACC ACA ACG</td>
</tr>
<tr>
<td>18</td>
<td>Tryptophan</td>
<td>Trp</td>
<td>TGG</td>
</tr>
<tr>
<td>19</td>
<td>Tyrosine</td>
<td>Tyr</td>
<td>TAT TAC</td>
</tr>
<tr>
<td>20</td>
<td>Valine</td>
<td>Val</td>
<td>GTT GTC GTA GTG</td>
</tr>
</tbody>
</table>
Visualize distribution of amino acid in a protein

- Given a gene sequence defining a protein
  \[ \text{TTCGGGAGCCTGGGCGTTACG...} \]
- Make histogram showing counts of amino acids that make up the protein

**Compute with text data!**
- Create \texttt{char} arrays
- Obtain subarrays (each a 3-letter codon)
- Search for and compare subarrays
- Do tally, draw histogram
Program sketch

- Given a dna sequence representing a protein

- For each codon (subvector of 3 chars)
  - Use codon dictionary to determine which amino acid the codon represents (get the 3-letter mnemonic)

- Tally the counts of the 20 amino acids

- Draw bar chart

See Insight §9.1. Here in lecture we extend the two functions for searching char arrays.
% dna sequence encoding protein
p = ['TTCGGGAGCCTGGGGCGTTACGTATGAAATGAAA' ... 
     'ATATGTACCAACGACCAATGACATTGAAAAAC'];
Program sketch

- Given a DNA sequence representing a protein
- For each codon (subvector of 3 chars)
  - Use codon dictionary to determine which amino acid the codon represents (get the 3-letter mnemonic)
- Tally the counts of the 20 amino acids
- Draw bar chart
% dna sequence encoding protein
p = ['TTCGGGAGCCTGGGCGTTACGTAAATGAAA' ...
     'ATATGTACCAACGACAATGACATTGAAAAC'];

for k = 1:3:length(p) - 2
    codon = p(k:k+2); % length 3 subvector
    % Search codon dictionary to find
    % the corresponding amino acid name
end

% Treat as an independent task
to be written as a function
function a = getMnemonic(s)
% s is length 3 row vector of chars
% If s is codon of an amino acid then
% a is the mnemonic of that amino acid

% Search for s in codon dictionary C
C= ['GCT Ala'; ...
   'GCC Ala'; ...
   'GCA Ala'; ...
   'GCG Ala'; ...
   'CGT Arg'; ...
   'CGC Arg'; ...
   'CGA Arg'; ...
   'CGG Arg'; ...
   'AGA Arg'; ...
   'AGG Arg'; ...
   'T' 'A' 'l' 'C' 'G' 'a'
   'C' 'A' 'l' 'C' 'G' 'a'
   'A' 'l' 'C' 'G' 'a'
   'G' 'C' 'T' 'A' 'l'
   'C' 'A' 'l' 'C' 'G'
   'G' 'C' 'G' 'A' 'l'
   'A' 'r' 'g'
   'A' 'r' 'g'
];

C is a 2-d array of chars
function a = getMnemonic(s)

% Given C, the 2-d char array dictionary
% Search it to find string s

r = 1;
while ~strcmp(s, C(r, 1:3))
    r = r + 1;
end

a = C(r, 5:7);

Assumes s is in C! If s not found → ERROR

Compares two char vectors. Returns true if they are identical; otherwise returns false.
function a = getMnemonic(s)
;
% Given C, the 2-d char array dictionary
% Search it to find string s

r= 1;
while ~strcmp(s, C(r, 1:3))
    r = r + 1;
end

a= C(r, 5:7);

Modify function so that gets empty char array if not found
function a = getMnemonic(s)

% Given C, the 2-d char array dictionary
% Search it to find string s

a = '';

nr = size(C, 1);

r = 1;

while r <= nr && ~strcmp(s, C(r, 1:3))
    r = r + 1;
end

if r <= nr
    a = C(r, 5:7);
end

If s not in C then a gets empty char array

See getMnemonic.m
% dna sequence encoding protein
p = ['TTCGGGAGCCTGGGCGTTACGTATGAATGAAA' ...
     'ATATGTACCAACGACAATGACATTGAAAAC'];

for k = 1:3:length(p)-2
    codon = p(k:k+2); % length 3 subvector
    mnem = getMnemonic(codon);
end
Given a dna sequence representing a protein

For each codon (subvector of 3 chars)
  - Use codon dictionary to determine which amino acid the codon represents (get the 3-letter mnemonic)

Tally the counts of the 20 amino acids

Draw bar chart
% dna sequence encoding protein
p = ['TTCGGGAGCCTGGGCGTTACGTGTTAATGAAA' ... 'ATATGTACCAACGACAATGACATTGAAAAC'];

for k = 1:3:length(p)-2
codon = p(k:k+2); % length 3 subvector
mnem = getMnemonic(codon);
% Tally: build histogram data
end
% dna sequence encoding protein
p = ['TTCGGGAGCCTGGGCGTTACGTAAATGAAA' ...
     'ATATGTACCAACGACAATGACATTGAAAAAC'];

count = zeros(1,20); % to store tallies

for k = 1:3:length(p)-2
codon = p(k:k+2); % length 3 subvector
mnem = getMnemonic(codon);
% Tally: build histogram data
ind = getAAIndex(mnem);
count(ind) = count(ind) + 1;
end

bar(1:20, count) % Draw bar chart