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 4 due Monday at 11pm EDT
- Consulting hours have resumed virtually
- Work with course staff to review Prelim 1. Now is the time to firm up any loose foundation!
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

\[ \text{W07629N4226} \]

meaning

\begin{align*}
\text{Longitude:} & \quad 76^\circ \, 29' \text{ West} \\
\text{Latitude:} & \quad 42^\circ \, 26' \text{ North}
\end{align*}

We may need to grab hold of the substring \text{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.

- Can have 2-d array of characters as well
  - A `2×6` matrix
  - A `1×13` row vector
A text sequence is a vector (of characters)

**Vectors**

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

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

- **: notation**
  
  \[ v = 2:5; \quad % \ v \ is \ [2 \ 3 \ 4 \ 5] \]

- **Appending**
  
  \[ v = [7 \ 0 \ 5]; \]
  
  \[ v(4) = 2; \quad % \ v \ is \ [7 \ 0 \ 5 \ 2] \]

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

**Strings**

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

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

- **: notation**
  
  \[ s = 'a':'g'; \quad % \ s \ is \ 'abcdefg' \]

- **Appending**
  
  \[ s = 'duck'; \]
  
  \[ s(5) = 's'; \quad % \ s \ is \ 'ducks' \]

- **Concatenation**
  
  \[ s = [s ' quack']; \]
  
  \[ % \ s \ is \ 'ducks quack' \]
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

a = 'CS1'
a = ['C','S','1']

b = [3 9]

c = uint8(b)

d = rand() > .5

a is a 1-d array with type char elements. Often called a string; NOT the same as a new type in Matlab 2017+ called string.

b is a 1-d array with type double elements. double is the default type for numbers in Matlab. We call b a “numeric array”

c is a 1-d array with type uint8 elements. We call c a “uint8 array”

d is a scalar of the type logical. We call d a “Boolean value”
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
A gene is a DNA fragment that codes for a protein, e.g.,

ATCGCTTTTGCACATTCTA...

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.,

ATCGCTTTGCGACATTCTA...

3-letter DNA “codons” identify the amino acid sequence that defines a protein
<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 CTA 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
  TTCGGGAGCCTGGGCGTTACG...
- Make histogram showing counts of amino acids that make up the protein

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

![Tally of Amino Acids in a Protein](image)
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 = ['TTCGGGAGCCTGGGCGTTACGTTAATGAAA' ...
     'ATATGTACCAACGACAATGACATTTGAAAAC'];

p is a 1-d array
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= ['TTCGGGAGCCTGGGCGTTACGTTAATGAAA' ... 
  'ATATGTACCAACGACAATGACATTGAAAC'];

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'; ...];

% C is a 2-d array of chars
C = ['G' 'C' 'T' ' ' 'A' 'l' 'a'
     'G' 'C' 'C' ' ' 'A' 'l' 'a'
     'G' 'C' 'A' ' ' 'A' 'l' 'a'
     'G' 'C' 'G' ' ' 'A' 'l' 'a'
     'C' 'G' 'T' ' ' 'A' 'r' 'g'
     'C' 'G' 'C' ' ' 'A' 'r' 'g']
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)) == false
    r = r + 1;
end

a = C(r, 5:7);

Compares two char vectors. Returns true if they are identical; otherwise returns false.

Assumes s is in C! If s not found → ERROR
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))==false
    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)) == 0
    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 = ['TTCGGGAGCCTGGGCGTTACGTTAATGAAA' ...
     '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
    mnem = getMnemonic(codon);
end
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 = ['TTCGGGAGCCTGGGCGTTACGTTAATGAAA' ...
     'ATATGTACCAACGACAATGACATTTGAAAAC'];

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 = ['TTCGGGAGCCTGGGCGTTACGTTAATGAAA' ...
     'ATATGTACCAACGACAATGACATTGAAAAC'];

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
function ind = getAAIndex(aa)
% Returns index of amino acid named by char vector aa.
% If aa does not name an amino acid, throw an error.

Display an error message and STOP program execution. (Not just a print statement.)
Use built-in function error.
See getAAIndex.m

Syntax: error( )
message to display
% dna sequence encoding protein
p = ['TTCGGGAGCCTGGGCGTTACGTTAATGAAA' ... 'ATATGTACCAACGACAATGACATTGAAAAC'];

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

See aminoAcidCounts.m
In addition to type `char`, we discussed ...

- Top-down design in program development—decompose the problem and then build the program one subproblem (one part, one refinement) at a time
- Search: Linear Search Algorithm

```plaintext
k = 1
while k is valid and item at k does not match search target
    k = k + 1
end
```
% Linear Search
% f is index of first occurrence
% of value x in vector v.
% f is -1 if x not found.

k = 1;
while k ≤ length(v) && v(k) ≠= x
    k = k + 1;
end

if k > length(v)
    f = -1; % signal for x not found
else
    f = k;
end

V = [12, 35, 33, 15, 42, 45]
X = 31
% Linear Search
% f is index of first occurrence
% of value x in vector v.
% f is -1 if x not found.

k = 1;
while k <= length(v) && v(k) ~= x
    k = k + 1;
end

if k > length(v)
    f = -1;  % signal for x not found
else
    f = k;
end

Suppose another vector is twice as long as v. The expected “effort” required to do a linear search is ...

A. squared
B. doubled
C. the same
D. halved