Previous Lecture:
- Characters arrays (type `char`)
- Review top-down design
- Linear search

Today’s Lecture:
- More on linear search
- Cell arrays
- Application of cell array: input from (and output to) a text file

Announcements:
- Discussion section in Zoom today/tomorrow
- Tutoring Wed-Mon (sign up on Canvas)
- Thurs lecture quizzes to be submitted via Gradescope
From last lecture: Linear Search

Search: Linear Search Algorithm

\[
\begin{align*}
k &= 1 \\
\text{while} & \quad \text{k is valid and item at k does not match search target} \\
& \quad \text{k} = \text{k} + 1 \\
\text{end}
\end{align*}
\]
% 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

See linearSearch.m, analyzeLinearSearch.m
Linear search:
Effort linearly proportional to length of vector searched

See linearSearch.m, analyzeLinearSearch.m
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: Special arrays where each component is a container for a collection of values
Limitations of primitive arrays

• Homogeneous data type
  • Can't represent tables

• Not nestable
  • No ragged arrays, lists-of-lists
  • Concatenation always "flattens"

• Multiple strings are awkward

```
'A' 'l' 'a' 'b' 'a' 'm' 'a' 
'N' 'e' 'w' 'Y' 'o' 'r' 'k' 
'U' 't' 'a' 'h' 'N' 'e' 'w' 'Y' 'o' 'r' 'k'
```

• `['John Doe', 33, true]`
  • Error using `horzcat`

• `[1, 2, 3; ... 4, 5]`
  • Error: Invalid expression.

• `[1, [2, 3], 4]`
  • `1 2 3 4`
New data type: Cell

- A cell's value may be of any type
  - Array of doubles
  - Array of characters
  - Array of more cells
- Each cell in an array may have a different type & size

Arrays of cells are still rectangular

```
  -4 -1
  'C' 'S'

  5 .91
  'M'
```

```
  'c' 'o' 'm'
  'T' 'E'
```

```
  1.1 -7
  12 8
```

```
  1.1 -1 12
```
Array vs. Cell Array

- **Simple array**
  - Each component stores **one scalar**. E.g., one `char`, one `double`, or one `uint8` value
  - All components have the same type

- **Cell array**
  - Each cell can store something “bigger” than one scalar, e.g., a vector, a matrix, a `char` vector
  - The cells may store items of different types
Application: lists of strings

• $C = \{ \text{'Alabama'}, \text{'New York'}, \text{'Utah'} \}$

<table>
<thead>
<tr>
<th>'Alabama'</th>
<th>'New York'</th>
<th>'Utah'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

• $C = \{ \text{'Alabama'}; \text{'New York'}; \text{'Utah'} \}$

<table>
<thead>
<tr>
<th>1</th>
<th>'Alabama'</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>'New York'</td>
</tr>
<tr>
<td>3</td>
<td>'Utah'</td>
</tr>
</tbody>
</table>

Compare with:

<table>
<thead>
<tr>
<th>1,</th>
<th>'A'</th>
<th>'l'</th>
<th>'a'</th>
<th>'b'</th>
<th>'a'</th>
<th>'m'</th>
<th>'a'</th>
<th>' '</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,</td>
<td>'N'</td>
<td>'e'</td>
<td>'w'</td>
<td>' '</td>
<td>'Y'</td>
<td>'o'</td>
<td>'r'</td>
<td>'k'</td>
</tr>
<tr>
<td>3,</td>
<td>'U'</td>
<td>'t'</td>
<td>'a'</td>
<td>'h'</td>
<td>' '</td>
<td>' '</td>
<td>' '</td>
<td>' '</td>
</tr>
</tbody>
</table>
### Use braces for creating & indexing cell arrays

<table>
<thead>
<tr>
<th>Primitive arrays</th>
<th>Cell arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create</strong></td>
<td><strong>Create</strong></td>
</tr>
<tr>
<td>[ 5, 4; ...</td>
<td>{ ones(2,2), 4; ...</td>
</tr>
<tr>
<td>1, 2; ...</td>
<td>'abc', ones(3,1); ...</td>
</tr>
<tr>
<td>0, 8 ]</td>
<td>9, 'a cell' }</td>
</tr>
<tr>
<td><strong>Index</strong></td>
<td><strong>Index</strong></td>
</tr>
<tr>
<td>( m(2,1) = \pi )</td>
<td>( C{2,1} = 'ABC' )</td>
</tr>
<tr>
<td>( \text{disp}(m(3,2)) )</td>
<td>( C{3,2} = \pi )</td>
</tr>
<tr>
<td></td>
<td>( \text{disp}(C{3,2}) )</td>
</tr>
</tbody>
</table>
Creating cell arrays

\[
C = \{ 'Oct', 30, \text{ones}(3,2) \};
\]

is the same as

\[
C = \text{cell}(1,3); \quad \% \text{optional}
C\{1\} = 'Oct';
C\{2\} = 30;
C\{3\} = \text{ones}(3,2);
\]

Can assign empty cell array

\[
D = \{};
\]

Comparison of bracket operators

• Square brackets \[
\]
• Create primitive array
• Concatenate (any) array contents
\[
[3 \ 1 \ 4 \ 1 \ [5 \ 9]] \\
[ 'a' \ {'b' \ ['c' \ 'd']}] \Rightarrow \{ 'a', 'b', 'cd' \}
\]

• Curly braces \{
• Create cell array enclosing contents
\[
\{ 3 \ [1 \ 4] \ 1 \ [5 \ 9] \} \\
\{ 'a' \ {'b' \ 'cd'} \}
\]
Example: Represent a deck of cards with a cell array

\[
\begin{align*}
D\{1\} & = \text{‘A Hearts’;} \\
D\{2\} & = \text{‘2 Hearts’;} \\
& \quad \vdots \\
D\{13\} & = \text{‘K Hearts’;} \\
D\{14\} & = \text{‘A Clubs’;} \\
& \quad \vdots \\
D\{52\} & = \text{‘K Diamonds’;} \\
\end{align*}
\]

But we don’t want to have to type all combinations of suits and ranks in creating the deck... How to proceed?
Make use of a suit array and a rank array …

\[
\text{suit} = \{\text{‘Hearts’, ‘Clubs’, …}
\]
\[
\text{‘Spades’, ‘Diamonds’}\};
\]

\[
\]
\[
\]

Then concatenate to get a card. E.g.,

\[
\text{str} = [\text{rank}\{3\} \text{ ‘ ‘ suit}\{2\}\};
\]

\[
\text{D}\{16\} = \text{str};
\]

So \(\text{D}\{16\}\) stores ‘3 Clubs’
To get all combinations, use nested loops

```matlab
suit= {'Hearts','Clubs','Spades','Diamonds'};
rank= {'A','2','3','4','5','6','7','8','9','10','J','Q','K'};
i= 1; % index of next card
for k= 1:4
    % Set up the cards in suit k
    for j= 1:13
        D{i}= [ rank{j} ' ' suit{k} ];
        i= i + 1;
    end
end
```

See function CardDeck
Example: deal a 12-card deck

<table>
<thead>
<tr>
<th></th>
<th>N:</th>
<th>E:</th>
<th>S:</th>
<th>W:</th>
</tr>
</thead>
<tbody>
<tr>
<td>D:</td>
<td><img src="deck.png" alt="Deck" /></td>
<td><img src="east.png" alt="East" /></td>
<td><img src="south.png" alt="South" /></td>
<td><img src="west.png" alt="West" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>N:</th>
<th>E:</th>
<th>S:</th>
<th>W:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,5,9</td>
<td>2,6,10</td>
<td>3,7,11</td>
<td>4,8,12</td>
<td></td>
</tr>
</tbody>
</table>

4k-3, 4k-2, 4k-1, 4k
% Deal a 52-card deck

N = cell(1,13); E = cell(1,13);
S = cell(1,13); W = cell(1,13);

for k=1:13
    N{k} = D{4*k-3};
    E{k} = D{4*k-2};
    S{k} = D{4*k-1};
    W{k} = D{4*k};
end
The “perfect shuffle” of a 12-card deck
Perfect Shuffle, Step 1: cut the deck

A B C D E F G H I J K L

A B C D E F
G H I J K L
Perfect Shuffle, Step 2: Alternate

A B C D E F G H I J K L

A B C D E F
G H I J K L

A G B H C I D J E K F L

1 2 3 4 5 6 7 8 9 10 11 12
Perfect Shuffle, Step 2: Alternate

A B C D E F G H I J K L

A B C D E F

G H I J K L

A G B H C I D J E K F L

2 4 6 8 10 12
Perfect Shuffle, Step 2: Alternate

A B C D E F G H I J K L

k
2k-1

A B C D E F

1 2 3 4 5 6

G H I J K L

1 3 5 7 9 11

A G B H C I D J E K F L

See function Shuffle
I want to put in the 3rd cell of cell array C a char row vector. Which is correct?

A. \(C\{3\} = \text{`a cat'}\);
B. \(C\{3\} = [\text{`a ' cat'}]\);
C. \(C(3) = \{\text{`a ' cat'}\}\);
D. Two answers above are correct
E. Answers A, B, C are all correct