Example: removing all occurrences of a character

From a genome bank we get a sequence

```
ATTG CCG TA  GCTA CGTACGC AACTGG
AAATGGC CGTAT...
```

First step is to “clean it up” by removing all the blanks. Write this function:

```matlab
function s = removeChar(c, s)
    % Return string s with all occurrences
    % of character c removed
```

Matrix vs. Cell Array

Vectors and matrices store values of the same type in all components.

A cell array is a special array whose individual components may contain different types of data.

Use braces `{}` for creating and addressing cell arrays.

```matlab
C = { 'Alabama','New York','Utah' }

1-d cell array of strings
```

```matlab
C {2,1} = 'ABC'
C {3,2} = pi
```

Cell Arrays of Strings

```
C = { 'Alabama','New York','Utah' }
```

1-d cell array of characters

```
C = { 'Alabama';'New York';'Utah' }
```

2-d array of characters

```
M = [ 'Alabama';
     'New York';
     'Utah' ]
```

```matlab
Matrix
```

```
m {1,1} = 3
m {1,2} = 4
m {2,1} = pi
m {2,2} = pi
```

```
Cell Array
```

```
C = { ones(2,2), 4
     'abc', ones(3,1)
     9 , 'a cell' }
```

```
C{2,1} = 'ABC'
C{3,2} = pi
disp(C{2,2})
```
Creating cell arrays...

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

is the same as

\[
\text{C} = \text{cell}(1,3); \quad \% \text{not necessary}
\]

\[
\text{C}(1) = \text{‘Oct’};
\]

\[
\text{C}(2) = 30;
\]

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

You can assign the empty cell array: \( \text{D} = {} \)

Example: Represent a deck of cards with a cell array

\[
\text{D}(1) = \text{‘A Hearts’};
\]

\[
\text{D}(2) = \text{‘2 Hearts’};
\]

\[
\vdots
\]

\[
\text{D}(13) = \text{‘K Hearts’};
\]

\[
\text{D}(14) = \text{‘A Clubs’};
\]

\[
\vdots
\]

\[
\text{D}(52) = \text{‘K Diamonds’};
\]

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...

\[
\text{suit} = \{ \text{‘Hearts’}, \text{‘Clubs’}, \ldots
\]

\[
\text{‘Spades’}, \text{‘Diamonds’} \};
\]

\[
\text{rank} = \{ \text{‘A’}, \text{‘2’}, \text{‘3’}, \text{‘4’}, \text{‘5’}, \text{‘6’}, \ldots
\]

\[
\text{‘7’}, \text{‘8’}, \text{‘9’}, \text{‘10’}, \text{‘J’}, \text{‘Q’}, \text{‘K’} \};
\]

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’

Because we don’t want to code all combinations of suits and ranks, we can create a suit array and a rank array...

Example: deal a 12-card deck

\[
\text{N}: 1, 5, 9 \quad \% 4k-3
\]

\[
\text{E}: 2, 6, 10 \quad \% 4k-2
\]

\[
\text{S}: 3, 7, 11 \quad \% 4k-1
\]

\[
\text{W}: 4, 8, 12 \quad \% 4k
\]

% Deal a 52-card deck

\[
\text{N} = \text{cell}(1,13); \quad \text{E} = \text{cell}(1,13);
\]

\[
\text{S} = \text{cell}(1,13); \quad \text{W} = \text{cell}(1,13);
\]

\[
\text{for } k = 1:13
\]

\[
\text{N}(k) = \text{D}(4*k-3);
\]

\[
\text{E}(k) = \text{D}(4*k-2);
\]

\[
\text{S}(k) = \text{D}(4*k-1);
\]

\[
\text{W}(k) = \text{D}(4*k);
\]

end

See function CardDeck

To get all combinations, use nested loops

\[
i = 1; \quad \% \text{index of next card}
\]

\[
\text{for } k = 1:4
\]

\[
\quad \% \text{Set up the cards in suit } k
\]

\[
\text{for } j = 1:13
\]

\[
\quad \text{D}(i) = [\text{rank}(j) \ ‘ ‘ \text{suit}(k) ];
\]

\[
\quad i = i+1;
\]

end

end

See function Deal
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
1 2 3 4 5 6
G H I J K L

A B C D E F
1 2 3 4 5 6
G H I J K L

Example: Build a cell array of Roman numerals for 1 to 3999

C(1) = ‘I’
C(2) = ‘II’
C(3) = ‘III’
: C(3999) = ‘MMMXXCIX’

Example

1904 = 1*1000 + 9*100 + 0*10 + 4*1
= M CM IV
= MCMIV

Concatenate entries from these cell arrays!
### Ones-Place Conversion

**Function**

```matlab
function r = Ones2R(x)
% x is an integer that satisfies
% 0 <= x <= 9
% r is the Roman numeral with value x.
Ones = {'I', 'II', 'III', 'IV', ...
        'V', 'VI', 'VII', 'VIII', 'IX'};
if x==0
    r = ''; % 0
else
    r = Ones{x}; % 1, 2, ..., 9
end
```

We want all the Roman Numerals from 1 to 3999. We have the functions Ones2R, Tens2R, Hund2R, Thou2R.

The code to generate all the Roman Numerals will include loops—nested loops. How many are needed?

- A: 2
- B: 4
- C: 6
- D: 8

### Similarly, we can implement these functions:

**Function**

```matlab
function r = Tens2R(x)
% x is an integer that satisfies
% 0 <= x <= 9
% r is the Roman numeral with value 10*x.
function r = Hund2R(x)
% x is an integer that satisfies
% 0 <= x <= 9
% r is the Roman numeral with value 100*x
function r = Thou2R(x)
% x is an integer that satisfies
% 0 <= x <= 3
% r is the Roman numeral with value 1000*x
```

Now we can build the Roman numeral cell array for 1,...,3999

```matlab
for a = 0:3 % possible values in thous place
    for b = 0:9 % values in hundreds place
        for c = 0:9 % values in tens place
            for d = 0:9 % values in ones place
                n = a*1000 + b*100 + c*10 + d;
                if n>0
                    C{n} = [Thou2R(a) Hund2R(b)...
                             Tens2R(c) Ones2R(d)];
                end
            end
        end
    end
end
```

Four strings concatenated together

The nth component of cell array C

### The reverse conversion problem

Given a Roman Numeral, compute its value. Assume cell array C(3999,1) available where:

- C{1} = ‘I’
- C{2} = ‘II’
- ...
- C{3999} = ‘MMMCMXCIX’

See script RN2Int

### Example: subset of clicker IDs

Find subset that begins with ‘h’

```matlab
L = {}; % Initialize empty cell array
for r=1:size(IDs,1) % Loop through each ID
    if IDs(r,1) == 'h' % Check if ID starts with 'h'
        L = [L, IDs(r,:)]; % Add ID to array L
    end
end
```

Concatenate cells or cell arrays—prone to problems!