Armstrong State University
Engineering Studies
MATLAB Marina – Characters and Strings Primer

Prerequisites

The Characters and Strings Primer assumes knowledge of the MATLAB IDE, MATLAB help, arithmetic operations, built in functions, scripts, variables, arrays, logic expressions, conditional structures, iteration, functions, and debugging. Material on these topics is covered in the MATLAB Marina Introduction to MATLAB module, MATLAB Marina Variables module, MATLAB Marina Arrays module, MATLAB Marina Logic Expressions module, MATLAB Marina Conditional Structures module, MATLAB Marina Iteration module, MATLAB Marina Functions module, and MATLAB Marina Debugging module.

Learning Objectives

1. Be able to use MATLAB's built in functions to read in and display numeric and string data.
2. Be able to convert characters to their equivalent ASCII value.
3. Be able to convert numeric data to strings for display.
4. Be able to format input and output using strings.
5. Be able to write MATLAB programs and functions that operate on strings.

Terms

character, string, ASCII, Unicode, type casting

MATLAB Functions, Keywords, and Operators

char, int8, uint8, strcmp, isspace, int2str, num2str, sprintf, fprintf, sscanf, fscanf, format

Characters

Characters are single lower or upper case letters, numbers, punctuation, or special characters. Characters are typically represented in computer systems using either ASCII or Unicode. The American Standard Code for Information Interchange (ASCII) uses 7-bits (8 bits for extended ASCII) to represent characters. It is commonly used to for transferring characters between input and output devices, for example from a computer keyboard to the computer. Unicode provides a unique coding for every character in almost every language and is platform and program independent. Unicode uses a 16-bit coding scheme to represent characters (4 hexadecimal digits). Since ASCII is a 7-bit code, up to 2^7 = 128 characters can be represented. Unicode can represent 2^{16} = 65536 characters. The first 128 codes (0000 to 007F) in Unicode correspond to the 128 ASCII codes. See http://unicode.org/ and http://www.unicode.org/charts/

MATLAB represents characters in ASCII. The ASCII code set can be found at the URL http://www.asciitable.com/. Character literals are specified in MATLAB using single quotes, for example `'a'`, `'!'`, `'7'`.

Strings

Strings have been used so far as text prompts with the MATLAB `input` function, as display and error messages with the MATLAB `disp` and `error` functions, and as labels and titles for plots. Strings are 1D arrays of characters (row vector of characters). String literals can be created by enclosing one or more characters in single quotes, for example: `'Dog'`, `'4 dogs and 1 cat'`, and `'4 – 6 = -2'`.

Since a string is a 1D array of characters, functions such as `length` and `size` and operations such as indexing can be performed on strings. Figure 1 shows examples of using the length function on strings and indexing strings.

```
>> name = 'Lucy the Dog';
>> length(name)
ans = 12
>> name(1:3)
ans = Luc
>> name(2:6)
ans = ucy t
```

Figure 1, Length and Indexing Strings

Typecasting

MATLAB's default data type is a real number. Casting allows one to convert data of one type to another type. Characters and strings can be cast to their ASCII equivalents, characters can be case to a single integer, and strings can be cast to an array of integers of the same length as the string.

The `char` function converts an array of positive integers to an array of characters. For example, `char(72)` is A, `char(105)` is i. Characters can also be cast to integers corresponding to their ASCII value with the `int8` or `uint8` operator. For example `int8('d')` returns 100 and `uint8('*')` returns 42. The `char` function can also be used to create a string, `char([65, 82, 34, 67])` returns AR"C. The `int8` and `unit8` operators can be used to convert a string to an array of integers. Figure 2 shows an example of using the `int8` operator to convert a string to an array of integers.

```
>> name = 'Lucy the Dog';
>> int8(name)
ans = 76  117   99  121   32  116  104  101   32  100  111  103
```

Figure 2, Casting a String to an Array of Integers

Casting is useful when we want to perform certain operations on characters and the desired operation or function is not be defined for characters.

Operating on Characters and Strings

Characters and strings can be concatenated (appended to each other) as vectors can. Figure 3 shows an example of string concatenation. MATLAB also has a built in `strcat` function for concatenating strings.

```
>> firstname = ['L','u','c','y']
firstname = Lucy
>> lastname = 'Dog';
>> name = [firstname, ' ', lastname]
name = Lucy Dog
>> name = [firstname, lastname]
name = LucyDog % same as before but without the separating space
```

Figure 3, String Concatenation

Arithmetic operations can be performed on strings but one must be careful when doing this. MATLAB will convert the string to a vector of integers (ASCII equivalent) and perform the arithmetic operation on the vector of integers yielding a vector of numbers as the result. For example, adding 1 to the string `'Lucydog'` adds one to the ASCII equivalent of each character in the string and results in a vector of numbers. The numbers (as long as they are integers in ASCII code range) can be cast to characters.

```
>> name = 'LucyDog'
name = LucyDog
>> name+1
ans = 77   118   100   122    69   112   104
>> char(name+1)
ans = MvdzEph
```

Figure 4, Adding One to each Element of a String

Remember that strings are arrays of characters. The same restrictions on arithmetic operations with vectors and arrays also apply to strings.

String Comparison

Strings can be compared using the same logical operators as for numbers >, <, >=, <=, = =, ~= (this is possible since MATLAB converts characters to their ASCII number), however, since strings are vectors the strings to be compared must be the same length (or one must have length 1). If strings of the same length are compared using the logical operators, the result is a vector of the same length with ones in the places where the characters in the strings match and zeros in the other places. If a string is compared to a character, the result is a vector of the same length as the string with ones in the places where the string elements match the character and zeros in the other places.

```
>> string1 = 'lucy';
>> string2 = 'Lucy';
>> string3 = 'dd';
>> string1 == string2
ans =    0     1     1     1
>> string1 == 'L'
ans =    0     0     0     0
>> string1 <= 'u'
ans =    1     1     1     0
>> string1 == string3
??? Error using ==> eq
Matrix dimensions must agree.
```

Figure 5, String Comparison using Logical Operators

MATLAB has a built in string compare function, `strcmp`, that allows one to compare two strings. The `strcmp` function returns true if the strings are identical (same length and composed of same characters) and false otherwise. MATLAB also has a built in function, `isspace`, to test if a character is a space. The `strcmp` function can also be used on cell arrays. Cell arrays will be covered later.

Arrays of Strings

Arrays of strings are 2D arrays of characters. MATLAB allows arrays of strings as long as each string in the array is the same length or the strings are padded with some character such as a space to make all of the strings the same length. The MATLAB statements in Figure 6a show two ways to create a 3 by 6 array of characters. Each row in the 2D array is a string of length six (a 1 by 6 array of characters).

```
>> arrayOfSstrings = ['Bob   ' ; 'Sally ' ; 'Sluggo'];
>> name1 = 'Bob   ';     % second way
>> name2 = 'Sally ';
>> name3 = 'Sluggo';
arrayOfSstrings(1,:) = name1;
arrayOfSstrings(2,:) = name2;
arrayOfSstrings(3,:) = name3;
```

Figure 6a, Array of Strings

The MATLAB statement in Figure 6b has a syntax error since the strings are not the same length and MATLAB will not allow creation of a 2D array with different row lengths. Padding each string to make the length 6 as was done in Figure 6a, allows the creating of an array of an array of three strings (3 by 6 array of characters).

```
>> arrayOfSstrings = ['Bob' ; 'Sally' ; 'Sluggo'];
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
```

Figure 6b, Error in Creating an Array of Strings

The character casting function `char` can be used to create arrays of strings. The `char` function when given a list of strings as arguments will pad each string with spaces to make all the strings the same length and then vertically concatenate them to create an array of strings.

```
>> arrayOfSstrings = char('Bob','Sally','Sluggo');
```

Figure 6c, Array of Strings

Formatted Input and Output

We have already seen that we can input strings from the user using the `input` function and display strings with the `disp` function. The `input` function takes a string, displays it in the command window, and takes an input from the user that can be assigned to a variable. By adding a second argument 's' after the text prompt the `input` function will automatically treat what is entered as a string (otherwise it attempts to evaluate the expression entered).

```
>> res = input('Enter a letter: ')
Enter a letter: a
??? Error using ==> input
Undefined function or variable 'a'.

>> res = input('Enter a letter: ','s')
Enter a letter: a
res = a

>> res = input('Enter an expression: ');
Enter an expression: 5 + 6
res = 11

>> res = input('Enter an expression: ','s');
Enter an expression: 5+6
res = 5+6
```

Figure 7, Reading Strings using `input` Function

In the first set of statements in Figure 7, the assignment expects the expression to evaluate to a number and when it does not, an error occurs. In the second set of statements in Figure 7, the `'s'` argument means the input is interpreted as a string and the string `'a'` is returned and assigned to the variable res. In the third set of statements in Figure 7, the entered expression `5+6` is evaluated and the numeric result is returned and assigned to the variable res. In the

fourth set of statements in Figure 7, the `'s'` argument means the input is interpreted as a string and the string '5+6' is returned and assigned to the variable res.

The `disp` function will display an array without displaying the array name. If the array is a string, the text in the string is displayed. MATLAB has other useful commands for formatting input and output, such as `fprintf`, `sprintf`, `fscanf`, `sscanf` , `int2str`, `num2str`, and `format`. The functions `int2str` and `num2str` are used to convert numbers to strings. They can then be appended onto another string and displayed.

The functions `sprintf` and `fprintf` allow one to format data and write it to either a string or a file using conversion specifications. The function `fprintf` can also be used to write data to the Command Window. The `fscanf` and `sscanf` allow one to input data using conversion specifications. When displaying formatted output in the Command Window, `fprintf` takes a format string and one or more pieces of data and displays the formatted data. The format string will contain one or more conversion specifications consisting of the % character, a conversion character, and additional flags/fields for certain conversion characters. The format string may also contain control characters indicated by the backslash character such as `\n` for a newline. The data to be formatted and substituted in place of the conversion specifications are provided in the same order as the specification and are separated by commas.

Figure 8a shows an example of using `fprintf` to display formatted output and Figure 8b shows an example using `int2str`, `num2str`, `sprintf`, and `disp` to display the same formatted output.

```
>> k = 2;
>> fprintf('Employee %d ',k);
Employee 2

>> fprintf('Bob Smith %d %d', 5, 5000);
Bob Smith 5 5000

>> fprintf('%s, Employee %d made $%-8.2f \n', 'Bob', 2, 572.45);
Bob, Employee 2 made $572.45
```

Figure 8a, Formatting and Displaying Output using `fprintf`

The conversion specification `%d` is for signed integers, `%s` is for strings, and `%f` is for fixed point numbers (real numbers). The parameters `-8.2` for the `%f` conversion specification indicate that the number should be left justified, displayed in a maximum of eight places, and have a precision of two places to left of decimal point. The full list of conversion specifications and control characters can be found in MATLAB's `fprintf` help.

```
>> k = 2;
>> disp(['Employee ' int2str(k)]);
Employee 2

>> k = 2;
>> displayText = sprintf('Employee %d ',k);
>> disp(displayText)
Employee 2

>> displayText = sprintf('%s, Employee %d made $%-8.2f \n', 'Bob',
2, 572.45);
>> disp(displayText)
Bob, Employee 2 made $572.45

>> k = 2;
>> name = 'Bob';
>> disp([name ', Employee ' int2str(k) ' made $' num2str(572.45)]);
Bob, Employee 2 made $572.45
```

Figure 8b, Formatting and Displaying Output using `int2str`, `num2str`, `sprintf`, and `disp`

The functions `fprintf` and `sprint` provide more flexibility and control formatting data for output and are their use is generally preferable to string concatenation. The `disp` function automatically generates a newline so the newline is often not needed when formatting a string for display using `disp`.

Convert Uppercase to Lowercase Function

Write a MATLAB function to convert all uppercase letters in a string to lowercase letters. Letters that are lowercase, punctuation, spaces, numbers, etc. should be left alone.

The convert only uppercase letters in a string to lowercase letters:

- The function will need the string to parse.
- The function will return the string with uppercase letters converted to lowercase letters.
- The function will need to determine for each character whether it is an uppercase or letter or not. This will require a conditional statement.
- If the character is an uppercase letter, the function will need to determine the equivalent lowercase letter and replace the uppercase letter with the lowercase letter. This will require adding 32 to the ASCII equivalent of the uppercase letter; casting the result to a character, and assigning the new character to the appropriate location of the string (same string index as the uppercase letter).
- The determination of uppercase and replacement if necessary must be done for all the characters in the string. This will require a loop that iterates over the characters in the string.

It will be helpful to start by developing a function that will work for a single character and then modifying it to work for a string (array of characters). The function should be tested for the following cases: empty string, single lowercase character, single uppercase character, single non-letter character, and a string with uppercase letters and lowercase letters and non-letters. Figures 9a, 9b, 9c, and 9d show the final `converttoLowercase` function that works for a string, the initial `converttoLowercase` function that works for a single character, the test program, and the output of the test program respectively.

```
function result = converttoLowercase(str)
% ------------------------------------------------------
% converttoLowercase converts uppercase letters in a string
% to lowercase
% ------------------------------------------------------
% Syntax: result = converttolowercase(str)
%  str is the string to convert
%  result is the string with letters all lowercase
% ------------------------------------------------------
% Notes: uppercase letters A to Z are ASCII 65 to 90
% lowercase letters a to z are ASCII 97 to 122
% ------------------------------------------------------
lengthString = length(str);
if (lengthString > 0)
    result = str;
    for k = 1:1:lengthString
        charValue = int8(result(k));
        if (charValue >= 65 && charValue <= 90)
            result(k) = char(charValue + 32);
        end
    end
else
    result = '';
end

end
```

Figure 9a, converttoLowercase Function

Note that in the test results the test case of an empty string yields an empty string so it is not seen in the display.

```
function result = converttoLowercase(ch)
result = ch;
charValue = int8(ch);
if (charValue >= 65 && charValue <= 90)
    result = char(charValue + 32);
end

end
```

Figure 9b, `converttoLowercase` Function for a Single Character

```
teststring1 = '';
teststring2 = 'a';
teststring3 = 'A';
teststring4 = '!';
teststring5 = 'Hello there Lucille.';
r = converttoLowercase(teststring1);
disp(r)
r = converttolowercase(teststring2);
disp(r)
r = converttoLowercase(teststring3);
disp(r)
r = converttoLowercase(teststring4);
disp(r)
r = converttoLowercase(teststring5);
disp(r)
```

Figure 9c, Test Program for `converttoLowercase` Function

```
a
a
!
hello there lucille.
>>
```

Figure 9d, Test Results for `converttolowercase` Function

Last modified Thursday, November 13, 2014