Chapter 6
Character Strings

Outline

6.1 Character String Concepts: Mapping and Casting

6.2 MATLAB® Implementation

6.3 Format Conversion Functions

6.4 Character String Operations

6.5 Arrays of Strings

Introduction

Individual characters have an internal numerical representation. The dominant representation is defined by the American Standard Code for Information Interchange (ASCII), where upper and lower case characters, numbers, and punctuation marks are represented by numbers between 0 and 127.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Strings of characters represent numerical values to the user: numerical values are stored in a special, internal representation for efficient numerical computation.

6.1 Character String Concepts: Mapping and Casting

We recognize that internally, any computer holds the values of variables as a series of ones and zeros. However, this is not really helpful to programmers.

Some languages require the programmer to specify how the content of each variable must be interpreted. Others, like MATLAB will infer the interpretation from the data stored in a variable.

- **Mapping** is how a program determines what the content of a particular variable means.

- **Casting** provide the programmer the ability to change the program's interpretation.

Mapping and Casting (continued)

For example, we already understand the data types **`double`** and **`logical`**.

`A = [2 4 6]` results in the machine believing that **`A`** contains data of type **`double`**.

`big = A > 3` results in the machine believing that **`B`** contains **`logical`** values. {MATLAB displays this as if it were numbers: **`[0 1 1].`**}

If we wanted to interpret **`big`** as **`double`** values, we would need to say:

`nums = double(big)`, thereby casting big to type **`double`**

Frequently, MATLAB reconsiders the current mapping of a variable depending on the operations being performed, but explicit casting is the safest way to preserve integrity in the data.

Mapping and Casting (continued)

If we issue the commands:

>> uint8('A')

The answer will be ans = 65 because in an 8-bit unsigned integer representation it has a value of 65.

>> char(100)

The answer will be ans = d because the 100^{th} element in the ASCII table is the 'd' character.

6.2 MATLAB® Implementation of Character Strings

Strings in MATLAB are defined by putting zero or more printable characters between single quote marks:

```
str = 'abcdefg'
```

- The variable **`str`** internally becomes a vector of length 7, one for each character.
- Externally, however, MATLAB assumes a mapping whereby the values of each vector element represent different printable characters according to the ASCII table, and makes the class of **`str`** to be **`char`**.
- To see the ASCII equivalents of each character, we cast **`str`** to **`double`**:
- **`ASCII_val = double(str)`** resulting in the values:

```
[97,98,99,100,101,102,103]
```

6.3 Format Conversion Functions

Converting from numbers to strings:

- **`int2str(x)`** returns a string assuming **`x`** is an integer
- **`num2str(x)`** returns a string assuming **`x`** has fractional parts
- **`sprintf(<fmt>, <params>)`** explicitly formats a string

Converting from strings to numbers:

- **`input(<prompt>)`** is a good way to convert automatically
- **`str2num(str)`** will convert a string representing well formatted numbers
- **`sscanf(str)`** formats a string into a number or vector/matrix of number

See MATLAB help files for details.

6.4 Character String Operations

Since a string is just a vector of ASCII values, any operations you can perform on a vector can be performed on a string. For example:

```
str = 'abcdefg'
str(1:2:end) -> 'aceg'
str > 'e' -> [0 0 0 0 0 1 1]
str+3 -> [100 101 102 103 104 105 106]
            % because of the addition
char(str + 'A' - 'a') -> 'ABCDEFG'
```

6.5 Arrays of Strings

- Arrays of strings can be built using the char(…) casting function, and uneven strings are padded with spaces.

```
char('abcde', 'cd', 'xyz') ->
    abcde
    cd
    xyz
```

- However, we usually collect strings in a Cell Array (see Ch 7)

Let’s Write some Code …

6.6 Engineering Example Encryption

As public access to information becomes more pervasive, there is increasing interest in the use of encryption to protect intellectual property and private communications from unauthorized access.

A *cryptosystem* is a way of encoding and decoding messages so that only certain people are able to read them. This case presents a cryptosystem based on matrix algebra and implemented using MATLAB. It is much more secure than simple systems you may have seen, such as replacement of each letter by a different letter.

6.6 Engineering Example Encryption

Encryption procedure

1. Translate the 15-character message to a 3 *× 5 matrix of ASCII* character codes.

2. Transform the matrix to an encrypted 3*×5 matrix of ASCII character* codes, by:

(a) subtracting 32 from each element of the matrix (so it is in 32-126 range)

(b) multiplying the matrix by a given 3 *× 3 matrix*

(c) reducing each matrix element to its remainder modulo 95

(d) adding 32 to each element of the resulting matrix (steps © and (d) are used to map the numbers to the 32-126 range).

3. Translate the encrypted 3*×5 matrix of ASCII character codes to* an encrypted 15-character message.

6.6 Engineering Example Encryption

Decryption procedure

1. Translate the 15-character encrypted message to a 3 *× 5 matrix* of ASCII character codes.

2. Transform the encrypted matrix back to the original 3*×5 matrix* of ASCII character codes, by:

(a) subtracting 32 from each element of the matrix

(b) multiplying the matrix by *the inverse of a given 3×3 matrix*

(c) reducing each matrix element to its remainder modulo 95

(d) adding 32 to each element of the resulting matrix (steps © and (d) are used to map the numbers to the 32-126 range).

3. Translate the original 3*×5 matrix of ASCII character codes back* to the original 15-character message.

6.6 Engineering Example Encryption

```
s = 'This is a test!'     % simple message
val = double(s);          % convert it into numbers
nume = reshape(val,3,5); % arrange array into a 3x5 matrix

% The transformation is affected by multiplying our matrix
% on the left by a 3x3 matrix.  To make the encryption possible
% we MUST:
% 'multiply by a 3x3 matrix whose entries are all integers
% and whose inverse has entries that are all integers'
m = [1 5 3; 2 11 8; 4 24 21];
% inv(m)=[39 -33 7;-10 9 -2;4 -4 1]

% Printable characters have ASCII codes in the range 32 to 126
% we must adjust the 3x5 message matrix by subtracting 32
nume = nume - 32;
% now num contains values in the range 0 to 84

% Apply the matrix transformation and adjust so that transformed
% message is in the ASCII range 32 to 126
ncoded = mod(m*(nume),95) + 32;

% convert these numbers to an array of characters
scoded = reshape(char(ncoded),1,15)
% so the encoded message is scode =]WQ1u\x5rYSM?Uy

% Decrypting a message requires to reverse the above steps to recover
% the original.
sdecoded = reshape(double(scoded),3,5);
ndecoded = mod(inv(m)*(sdecoded-32),95) + 32;
sdecoded = reshape(char(ndecoded),1,15)
```

6.6 Engineering Example Encryption

```
>> cryptosystem

s =

This is a test!


scoded =

]WQ1u\x5rYSM?Uy


sdecoded =

This is a test!

>>
```

Homework on Chapter 6 is posted on the website:

http://www.ee.nmt.edu/~erives/289_F12/EE289.html

Homework is due within a week