Character Encoding

CS 8: Introduction to Computer Science, Winter 2019
Lecture #14

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Administrative

- **LAST HOMEWORK! Hw08:** Due Wednesday 3/13
  - It's a short one

- **Hw07:** Due Monday 3/11
- **Lab07:** Due Monday 3/11
- **Project1**: Due Thursday 3/14

Lecture Outline

- ASCII Codes, UTF Codes
- Functions **ord()** and **chr()**
- Exercises

ASCII TABLE

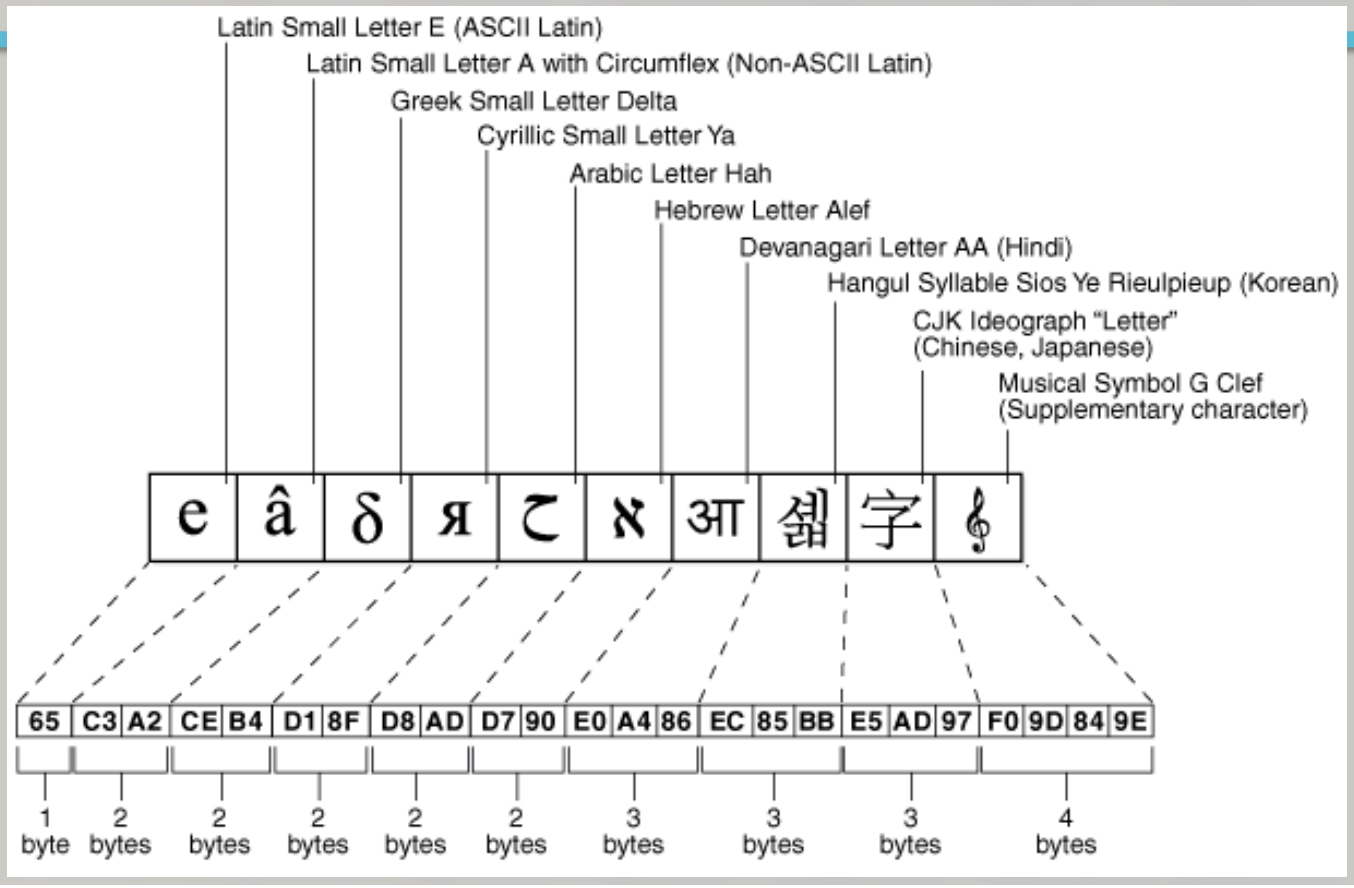
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

UTF Codes

Unicode Transformation Format

- ASCII uses 7 bits for its codes
  - This means there are 2^7 (or 126) possible codes
  - Preferred encoding for basic text files in the Latin alphabet
- UTF-8 is another standard
  - Uses 8 bits for its codes (so, 2^8 = 256 possibles)
  - Backwards compatible with ASCII
  - Preferred encoding for e-mail and web pages
- UTF-16 is the “widest” standard (uses 16 bits)
  - Capable of encoding the entire Unicode repertoire.

UTF-8 Schemes



Functions `chr(n)` and `ord(c)`

- **Characters** are stored as **numbers** in computer memory
  - There are standard codes for characters, e.g. **ASCII**, **UTF-8**, etc…
- For example, 'A' has code 65 in ASCII
  - Use the ord function to verify this: ord('A') is 65
  - Notice 'A' is not same as 'a': ord('a') is 97
- Every character, **seen** (e.g. %, !, G, =, space, tab,…) and **unseen** (e.g. CONTROL-X, newline…) has an ASCII code

Functions `chr(n)` and `ord(c)`

- Likewise, you can find character associated with a particular code using chr function, for example:

  chr(65) is 'A'

- You can manipulate numbers in order to process characters

  chr(ord('a') + 3) is chr(97), which is 'd'

- Notice **digit *characters*** have codes too!

  ord('6') is 54

Examples

- How can I find out what's 13 letters *after* '**e**'??
  - *Easy answer*: recite the alphabet from 'e' and count 13 places
  - *Code answer*: `chr( ord('e') + 13 )`, which is `'r'`

- How can I find out what's 19 letters *before* '**Z**'??
  - *Code answer*: `chr( ord('Z') - 19)`, which is `'G'`

- What's the ASCII code for the hashtag character??
  - *Code answer*: `ord('#')`, which is `35`

Harder Example…

- How can I do an (not-found-in-Python)
  "addition" of 2 numeral strings, like **'3'** and **'4'** and get **'7'**??

- First ask: how can I make '3' into 3? (***HINT***: We'll need a baseline…)
- That baseline is **ord('0')** --- how far away in the ASCII is '3' from '0'???
- Note that: ord('3') – **ord('0')** = 3
- So the "addition" is done like this:

(ord('3') – ord('0')) + (ord('4') – ord('0')) = 7 (an int)

or, ord('3') + ord('4') – 2*ord('0') = 7

So I Can Create a Function to do This!

```
def addChars1(char1, char2):
    numAddASCII = ord(char1) +  ord(char2) – 2*ord('0')
    return numAddASCII      # Returns an integer
```

Important Caveat!

Only works with 1 character numbers!

What if I Wanted to Return a String Result?

```
def addChars2(char1, char2):
    numAddASCII = ord(char1) +  ord(char2) – 2*ord('0')
    charNum = chr(numAddASCII + ord('0'))
    return charNum      # Returns a string
```

Important Caveat!

Again, only works with 1 character numbers!

Exercise 1

- Create a function **MyCipher(myStr)** – takes a string argument
- Makes every letter become the letter after it
  - Letter ‘a’ becomes ‘b’, ‘b’ becomes ‘c’, etc…
  - So that “**hello**” becomes “**ifmmp**” (*encryption*)
- How would you *decrypt* this?

MyCipher() and its Reverse

```
def MyCipher(myStr):
    enc_str = ''
    for c in myStr:
        enc_str += chr(ord(c) + 1)
    return enc_str
```

```
def ReverseMyCipher(myStr):
    dec_str = ''
    for c in myStr:
        dec_str += chr(ord(c) - 1)
    return dec_str
```

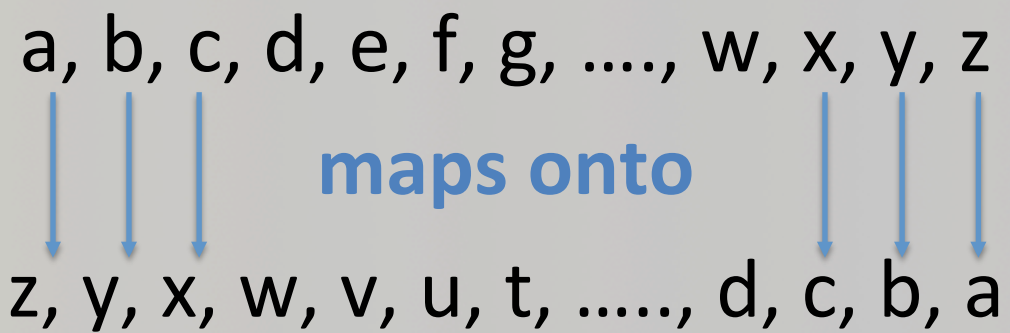
Exercise 2

Mirrored Alphabet *(or "the first shall be the last")*

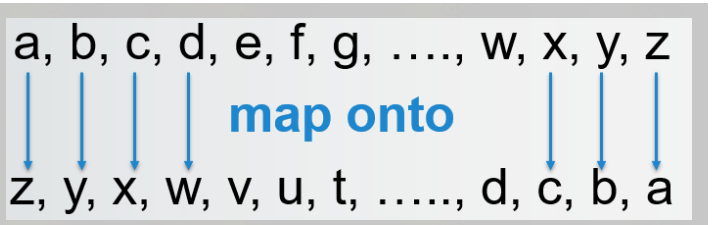
- The letters a, b, c, d, … w, x, y, z map onto
  z, y, x, w, … d, c, b, a
- So that "**bye**" becomes "**ybv**"
  and "**maria**" becomes "**nzirz**"
  and "**abcdef**" becomes "**zyxwvu**"
- How would you decrypt this?
- Would you say this is a *symmetric encryption scheme*?

Mirrored Alphabet Cipher

- Let's examine the thinking behind this:



Our Algorithm



1. Given a string (message) with N number of letters
2. Go thru every letter in order to examine it (*how?*)
3. Apply "mapping formula" to each letter

   (*don't know what that "formula" is yet, but that's ok…*)
4. Once formula is applied,

   "gather up the new letters" into a NEW string (*how?*)
5. Return that NEW string as the encoded message

MirrorEncrypt()

```
def MirrorEncrypt(message): # message is a string type
    result = ''             # start with an empty result
    for c in message:       # go thru every letter in message
                            # let's apply the "mirror" formula:
        nc = ord(c)
        nr = ord('a') + ord('z') - nc

       # then accumulate the encoded chars, one at a time
        result = result + chr(nr)
    return result
```

MirrorEncrypt() Questions

- What happens if I try `MirrorEncrypt(MirrorEncrypt("cat"))`?
  - *Why*?

- What happens if I try `MirrorEncrypt("CAT")`?
  - *Why*?

YOUR TO-DOs

- ❑ **HW7** (due on **Monday, 3/11**)
- ❑ **HW8** (due on **Wednesday, 3/13**)
- ❑ **Lab7** (due on **Monday, 3/11**)
- ❑ **Project Assignment** (due on **Thursday, 3/14**)

</LECTURE>