Lecture 3: Binary wrap-up & strings

CSE 29: Systems Programming and Software Tools

Olivia Weng

Announcements

- Exams
  - Only 1 makeup exam
  - Sign up for a practice session at the CBTF on prairietest.com

- Discussion is optional

- Problem set 1 will be released today

- Go to lab today in CSE B250!

Review: Two's complement

- Signed values in C are represented as two's complement
  - Lets us represent both positive and negative values
  - Example data types: `char`, `int`, `int8_t`

1-byte signed

↳ 4 bytes signed

1-byte signed

- Unsigned values in C only represent values >= 0
  - Example data types: `unsigned char`, `unsigned int`, `uint8_t`

1-byte unsigned

char : [-128, 127]

unsigned char : [0, 255]

} = 256 values

Review: $\overbrace{\text{Hexa}}^{6} + \overbrace{\text{decimal}}^{10} = 16$

- Long binary representations is hard for humans to read
- Hexadecimal helps humans read binary
  - Hexadecimal = base 16 ~ 16 values : [0-15]
  - Decimal = base 10 — 10 values : [0 - 9]
  - Binary = base 2 ~ 2 values : [0, 1]

$00000000 = \underbrace{0x}_{\text{prefix}}00$

||

I am hex

Intro to Hexadecimal

- Hexadecimal = 16 values

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

What is this hexadecimal number in binary?

- 0xB3 → 11 = $\overset{B}{1011}\ \overset{3}{0011}$

- 0x5A = $\underbrace{0101}_{5}\ \underbrace{1010}_{A}$

What is this binary number in hexadecimal?

11 0 11 10 : Decimal

- 1011 0000 1011 1010 : binary

0x B Ø BA : Hex

printf("%x", 0);

Strings in C

What is a string in C?

- String is an array of characters
- String is terminated when it encounters a null character

No special String type in C

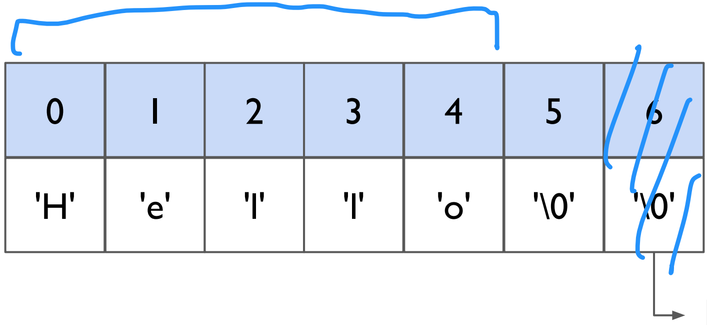
char str[]

'\0' == 0

A string is an array of characters

```
char hello[7] = "Hello";
```

6



0	1	2	3	4	5	~~6~~
'H'	'e'	'l'	'l'	'o'	'\0'	~~'\0'~~

NULL terminator= 0

What if there is no NULL char?

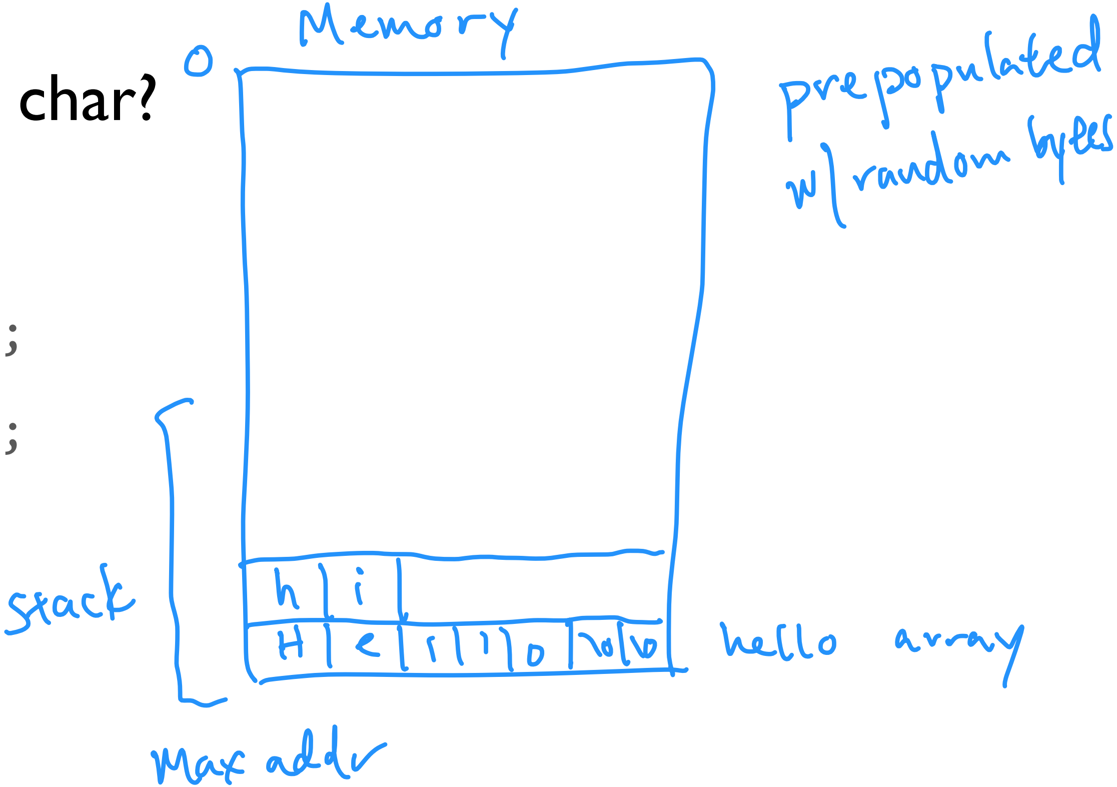
- What will be printed?

```
char hello[7] = "Hello";

char hi[2] = {'h', 'i'};

puts(hello);

puts(hi);
```



What if there is no NULL char?

- C will do exactly what you tell it to do

```
char hello[7] = "Hello";

char hi[2] = {'h', 'i'};
```

0	1	0	1	2	3	4	5	6
'h'	'i'	'H'	'e'	'l'	'l'	'o'	'\0'	'\0'

- All variables share the same linear memory space!!

char array vs String

- You can still declare an array of char
- A C string is specifically a char array that ends in a NULL terminator
    - When printed with %s, the elements of the array will be interpreted as ASCII

```
// stores array of type char - signed 1-byte values
char numbers[3] = {1, 2, 3};

char letters[3] = {'h', 'i', '\0'};
char letters2[3] = {104, 105, 0}; // '\0' == 0

printf("%s\n", letters);
printf("%s\n", letters2);
```

How to get the length of a string?

- Use `strlen()`!

```
#include <stdio.h>
#include <string.h>

char letters[3] = {'h', 'i', '\0'};
char letters2[3] = {104, 105, 0}; // '\0' == 0

printf("%s len = %d\n", letters, strlen(letters));
printf("%s len = %d\n", letters2, strlen(letters2));
```

char datatype

- char = 1 byte
  - equivalent to int8_t

- Store human readable English characters in char

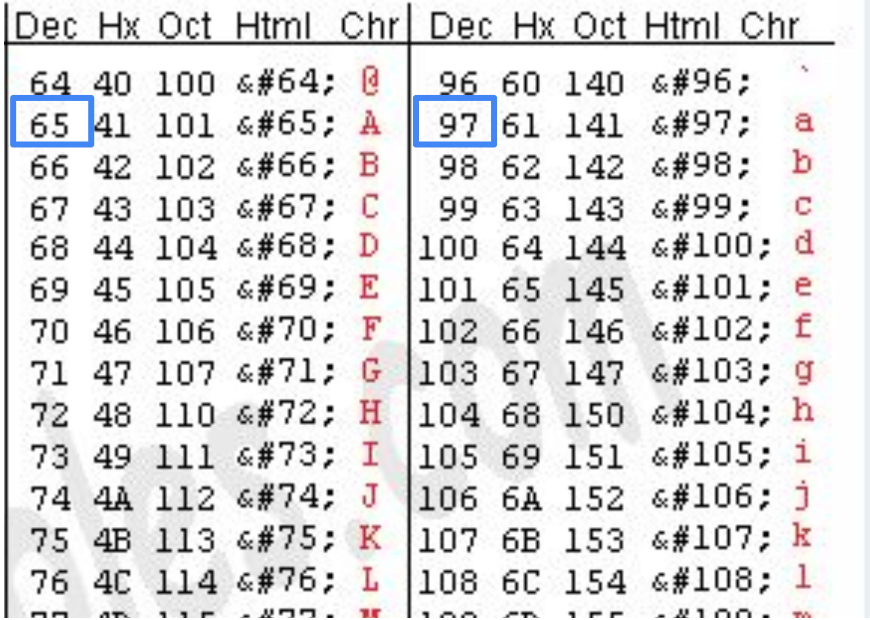
- ASCII: The English characters have number equivalents
  - 0-127 encodes English characters

How can we go from uppercase to lowercase?

- Demo:

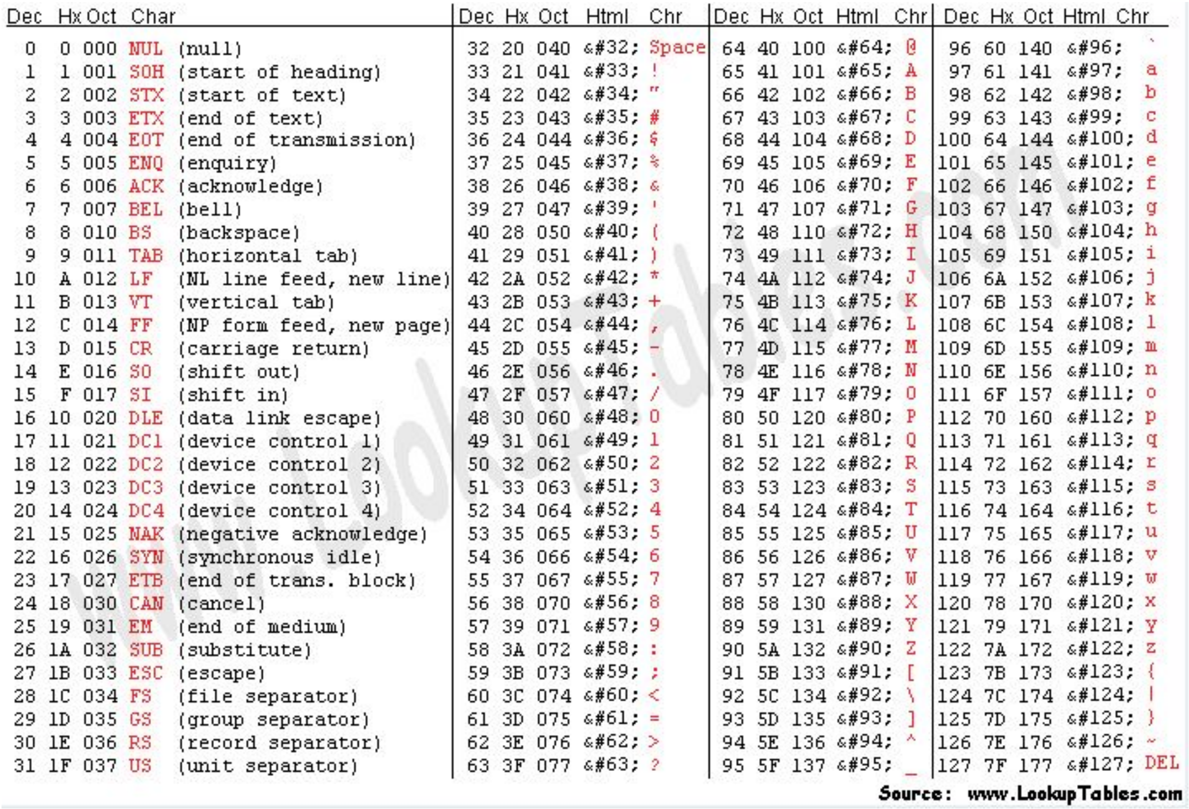
```
char to_lower(char c);
```

'A' = 65
+32
97 = 'a'



Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
64	40	100	&#64;	@	96	60	140	&#96;	`
65	41	101	&#65;	A	97	61	141	&#97;	a
66	42	102	&#66;	B	98	62	142	&#98;	b
67	43	103	&#67;	C	99	63	143	&#99;	c
68	44	104	&#68;	D	100	64	144	&#100;	d
69	45	105	&#69;	E	101	65	145	&#101;	e
70	46	106	&#70;	F	102	66	146	&#102;	f
71	47	107	&#71;	G	103	67	147	&#103;	g
72	48	110	&#72;	H	104	68	150	&#104;	h
73	49	111	&#73;	I	105	69	151	&#105;	i
74	4A	112	&#74;	J	106	6A	152	&#106;	j
75	4B	113	&#75;	K	107	6B	153	&#107;	k
76	4C	114	&#76;	L	108	6C	154	&#108;	l
77	4D	115	&#77;	M	109	6D	155	&#109;	m

ASCII Table



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

Source: www.LookupTables.com

Demo

```
is_ascii();


int32_t capitalize_ascii(char str[]);
// Returns the number of characters capitalized and capitalizes
the lowercase
// a-z ASCII characters of str in-place.
```

$\text{"H}\underbrace{\text{ello}}\text{"} \rightarrow \text{"HELLO"}$

return 4

What about non-English characters?

What about non-English characters?

- Thousands more characters used in languages around the world
- ASCII does not define:
    - Spanish: é
    - Chinese: 中
    - Emoji: 🐪
- char datatype of 1 byte only encodes 256 possible bit patterns
- Challenge: Millions of lines of code written that assumed 1 byte ASCII chars

UTF-8: Unicode encoding

- Use more bits to encode more characters!
- Code point: an integer representing a character (e.g., 'A' == 65)

- Normal ASCII code point: Highest order bit of byte is 0xxxxxxx
  - UTF-8 is backwards compatible with ASCII!
- Multi-byte code point: Highest order bit of byte is 1xxxxxxx