

Lecture 4: Hexadecimal & strings

CMPE 120: Computer Organization & Architecture

Olivia Weng

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

Intro to Hexadecimal

- Long binary representations is hard for humans to read
- Hexadecimal helps humans read binary
 - Hexadecimal = base 16
 - Decimal = base 10
 - Binary = base 2

Intro to Hexadecimal

- Hexadecimal = 16 values

[illegible]

What is this hexadecimal number in binary?

- 0xB3

- 0x5A

What is this binary number in hexadecimal?

- 1011 0000 1011 1010

Booleans

- No boolean data type in C
- All **non-zero** values evaluate to **true**
- Only **zero** evaluates to **false**

Strings in C

What is a `string` in C?

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

A string is an **array** of characters

```
char hello[6] = "Hello";
```

0	1	2	3	4	5
'H'	'e'	'l'	'l'	'o'	'\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[6] = "Hello";
```

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

0	1	0	1	2	3	4	5
'h'	'i'	'H'	'e'	'l'	'l'	'o'	'\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);
```

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

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	 	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		DEL

Demo

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