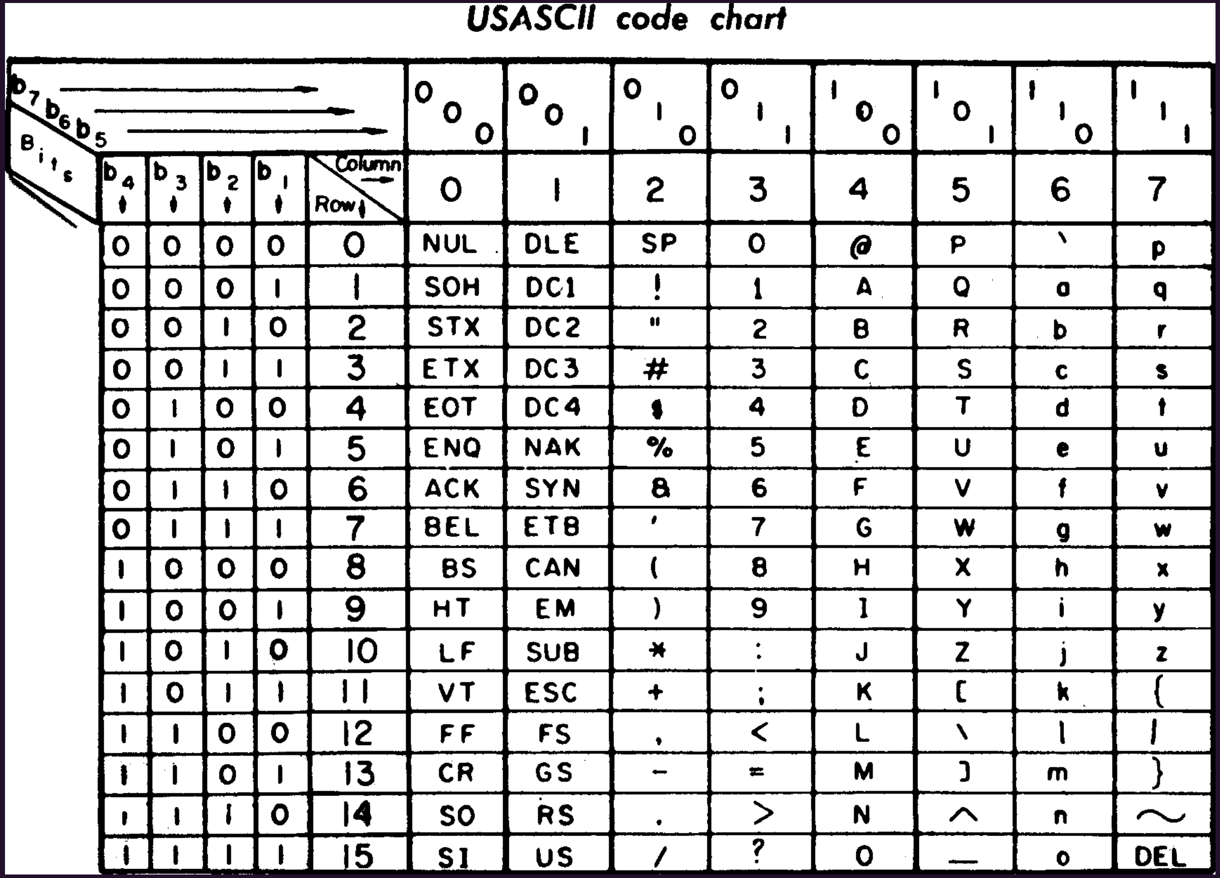
ASCII !

ASCII

- American Standard Code for Information Interchange
  - Work began in 1960
  - Standardized in 1963
- 128 characters in low 7 bits of a byte
  - 95 printable characters
  - 33 non-printable *control* codes
    - **`'\n'`** is **`LF`** - Line Feed - "*The action of advancing paper in a printing machine by the space of one line.*"
    - **`Ctrl+D`** emits **`EOT`** - End of Transmission

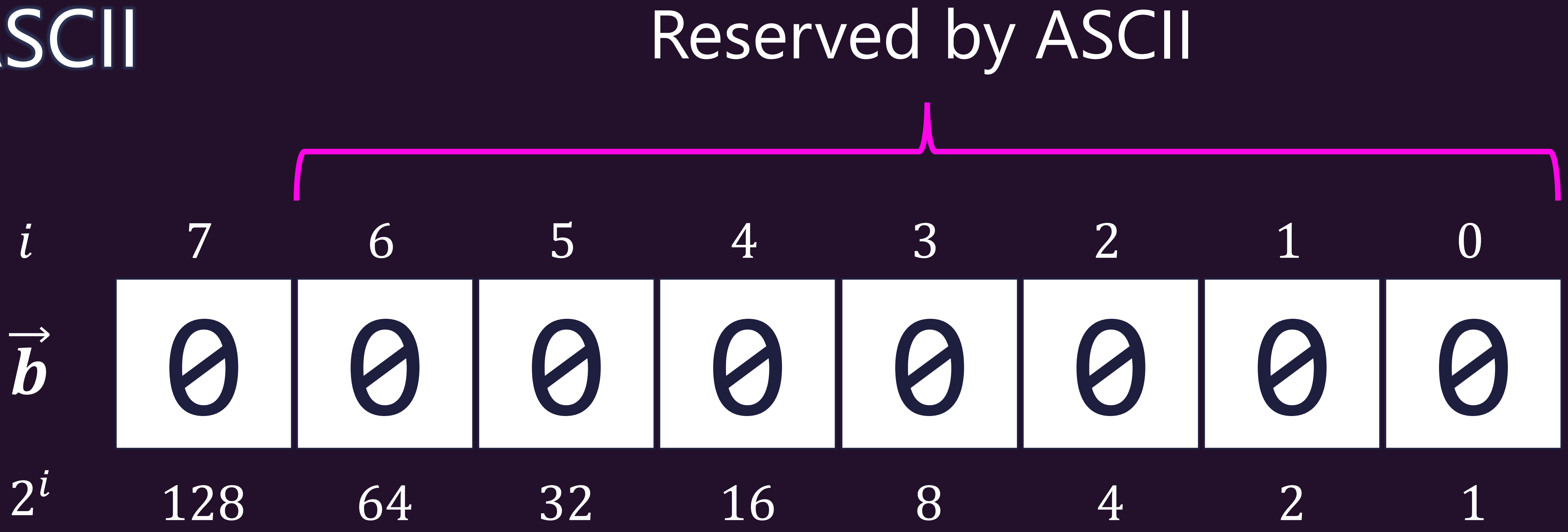


USASCII code chart

Bits b7 b6 b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4 ↓	b3 ↓	b2 ↓	b1 ↓	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Source: https://en.wikipedia.org/wiki/ASCII

ASCII



Reserved by ASCII

i	7	6	5	4	3	2	1	0
$\vec{b}$	0	0	0	0	0	0	0	0
2^i	128	64	32	16	8	4	2	1

"The committee voted to use a seven-bit code to minimize costs associated with data transmission. Since perforated tape at the time could record eight bits in one position, it also allowed for a *parity bit* for *error checking* if desired."
Source: https://en.wikipedia.org/wiki/ASCII

C Strings are *null terminated* char arrays

- Null character `'\0'` is a byte with a 0 value

- Thus, the length of a string literal is always # of chars + 1 for null termination character.

- The memory representation of a C string is *only* its char array.
  - In most higher-level languages, e.g. Java, a string's *length* is also stored alongside the *char* array.
  - So how would you find the length of a C "string"?

	Contents$_2$	Contents$_{16}$	Contents$_{10}$	Contents$_C$
F	00000000	00	0	'\0'
E	00001010	0A	10	'\n'
D	00100001	21	33	'?'
C	00111111	3F	63	'!'
B	01100100	64	100	'd'
A	01101100	6C	108	'l'
9	01110010	72	114	'r'
8	01101111	6F	111	'o'
7	01010111	57	87	'W'
6	00100000	20	32	' '
5	00101100	2C	44	','
4	01101111	6F	111	'o'
3	01101100	6C	108	'l'
2	01101100	6C	108	'l'
1	01100101	65	101	'e'
0	01001000	48	72	'H'

In C, a variable's address is its first, lowest addressed byte in memory.

- Arrays arrange for the 0th index to be the lowest address. Why?
- Because finding the addresses of other indices is easier arithmetic!

The example to the right illustrates how the string literal **`"Hello, world?!\n"`** would be represented if stored at memory address 0.

Notice it would be ***the exact same in memory as***:

```
char a[16]   = { 72, 101, 108, 108, 111, 44, 32, 87,
                111, 114, 108, 100,  63, 33, 10, 0 }

uint8_t b[16]= {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,
               0x6F,0x72,0x6C,0x64,0x3F,0x21,0x0A,0x00};
```

Address	Contents$_2$	Contents$_{16}$	Contents$_{10}$	Contents$_C$
F	00000000	00	0	'\0'
E	00001010	0A	10	'\n'
D	00100001	21	33	'?'
C	00111111	3F	63	'!'
B	01100100	64	100	'd'
A	01101100	6C	108	'l'
9	01110010	72	114	'r'
8	01101111	6F	111	'o'
7	01010111	57	87	'W'
6	00100000	20	32	' '
5	00101100	2C	44	','
4	01101111	6F	111	'o'
3	01101100	6C	108	'l'
2	01101100	6C	108	'l'
1	01100101	65	101	'e'
0	01001000	48	72	'H'