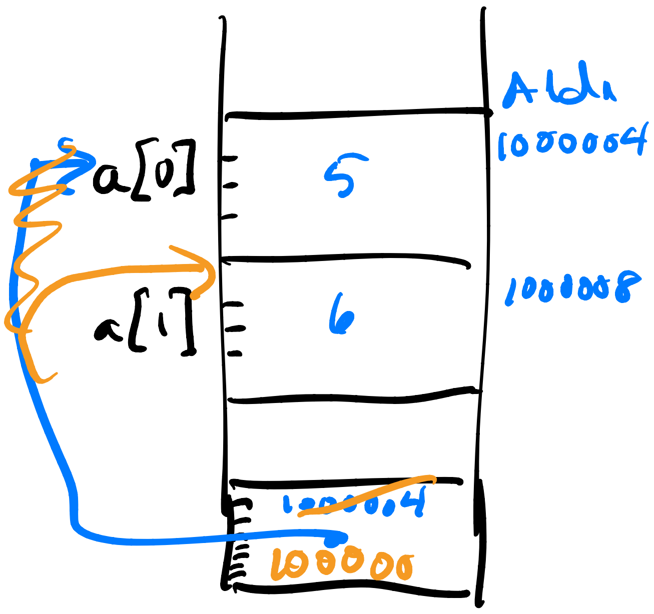
CS208

W, 28 Jan 2026

Pointer vars in C

++ ⇝ advances the ptr by sizeof the thing being pointed to



```
int a[2] = {5, 6};
int *p = a;
p++;
```

```
int a[8];
int *p1 = a;
int *p2 = a;
p2++;
p2++;
p2++;
p2++;
```

p2 - p1

compiler treats this as a "how many ints apart are these addresses"

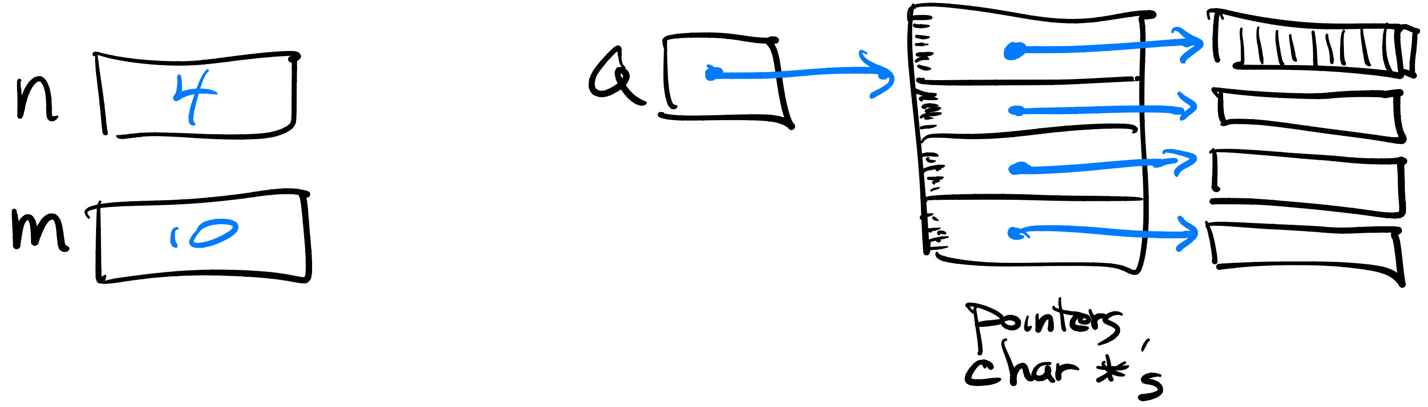
$$\left(\left(\text{address in } p2\right) - \left(\text{addr in } p1\right)\right) / \text{sizeof(int)}$$

$16/4 = \boxed{4}$

$$\underbrace{(\text{long})(p2)}_{\uparrow} - \underbrace{(\text{long})(p1)}_{\downarrow 16}$$

"Hey compiler, stop thinking of p2 as a ptr + think of it as an integer"

Worksheet, function f



Have to free stuff that succeeded if a later malloc fails + you want to return NULL.

Quiz #7.

```
char *pointers[3]

pointers[0] == 0x000000000000803e
        [1] == 0x0 ———————— 03a
        [2] == 0x0 ———————— 009
```

What's the type of pointers[0]? char *

```
printf("%s", pointers[0]);
```

Quiz #3

```
char s1[6];
```

Symbol s1 can be used as "the address of the array" = "the address of s1[0]"

```
printf("%s", s1);
```

$\rightsquigarrow$ bit (stopped at \0)

wants char

char*

```
printf("%c", s1)
```

$\leftarrow$ does not compile

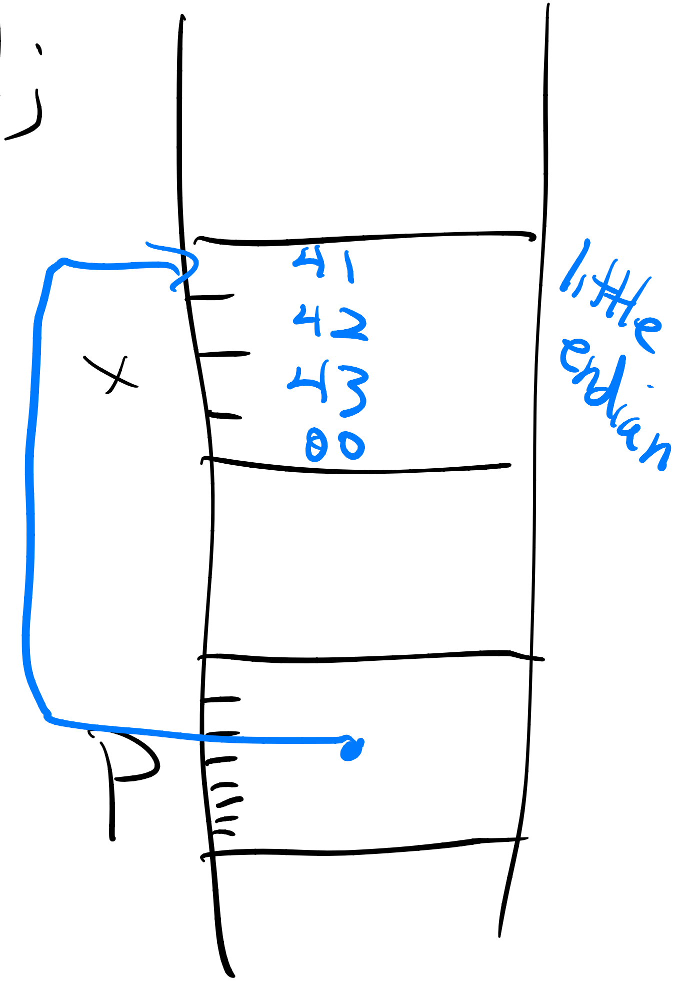
```
printf("%c", s1[2]);
```

$\rightsquigarrow$ t

```
int x = 0x00434241;
char *p = (char*)(&x);
printf("%s", p);
```

↓

ABC



41
42
43
00
x
p
little endian