Objectives - Tue 2/5/2019

- Code review
  - DRY Rule ("Don't' Repeat Yourself") → deduping code
  - Refactoring to obviate need for flag
  - Coding in paragraphs
  - Effective commenting
  - CQ: Initialize where you declare
  - CQ: `bool`/`true`/`false` for flags… and what to name them
  - CQ: naming functions, variables, etc.
- Dynamic memory
  - `malloc(…)`   `free(…)`
  - don't forget the *! (`int* array = malloc(n * sizeof(*array);`)
  - CQ: Free where you `malloc(…)`
  - CQ: No type cast on `malloc(…)`
  - CQ: `sizeof(`*EXPRESSION*`)` ✓   … not `sizeof(`*TYPE*`)` ✗

How to name a bool

is_empty

has_color

needs_remainder

have_more_digits

For grammar nerds only: third person simple present verb phrase

Does it make sense in if? if(is_empty){...}

Yes, it ...

is_empty ...
needs_remainder

Yes, we ...

have_more_digits
should_create_file ...

for vs. while

PREFERRED

```
for ( init ; condition ; next ) {
    body
}
```

OK WHEN MORE READABLE

```
init
while (condition) {
    body
    next
}
```

same

Prefer for unless while is truly more readable/maintainable.

(segoe to dynamic memory)

VLAs

Do not use in 264!!!

C89	C99
not supported	can be disabled with -Wvla

Makes code ~~very~~ hard to debug

dynamic memory in C

$$\text{malloc}\left(\underbrace{\boxed{\qquad\qquad\qquad}}_{\text{number of bytes to allocate}}\right)$$

returns a void*
address of anything

$$\text{malloc}\left(\underbrace{\boxed{\qquad}}_{\text{\# of objects}} * \text{sizeof}\left(\underbrace{\boxed{\qquad\qquad}}_{\text{expression}}\right)\right)$$

$\text{sizeof}\left(\underbrace{\boxed{}}_{\text{type}}\right)$ ✗

$\text{sizeof}\left(\underbrace{\boxed{}}_{\text{expression}}\right)$ ✓

$\uparrow$

$\Rightarrow \text{sizeof}\left(\begin{array}{l}\text{type of}\\ \text{that expr}\end{array}\right)$

$\underset{✓}{\text{sizeof}(5)} \iff \underset{✗}{\text{sizeof}(\text{int})}$

*(On our platform)

```
sizeof(1000000) == 4
sizeof(0)       == 4
int* a_n = &n;
sizeof(a_n) == 8
sizeof(&n)  == 8
sizeof(*&n) == 4
       weird!
X sizeof(int*) == 8 ← NO!
```

Common usage of malloc(...)

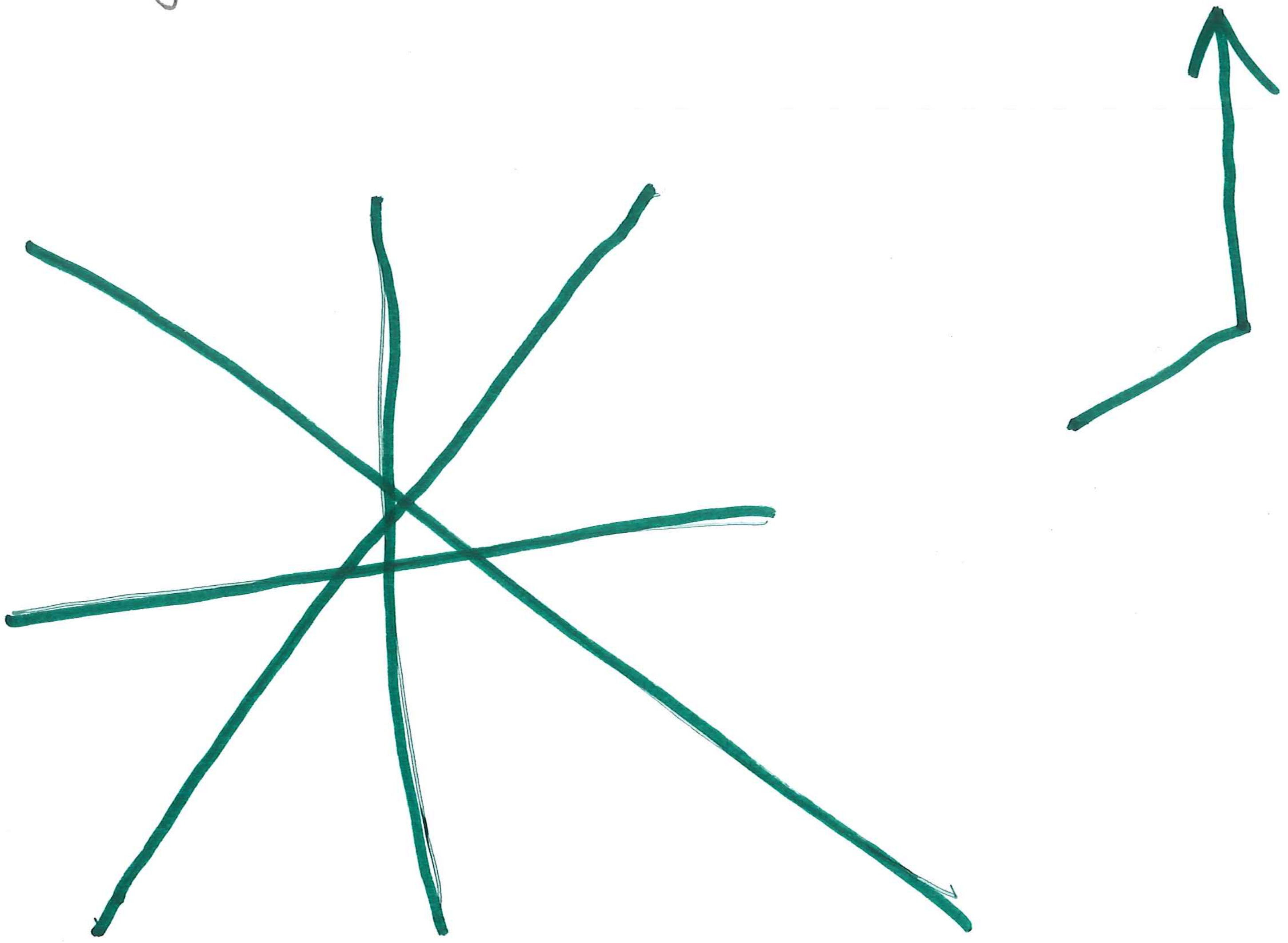
```
int* array = malloc(n * sizeof(*array));
```

type of	array is	int*
	*array	int

⚠ Don't forget the *

Don't forget the * !!!

```
int* array = sizeof(num-elements * sizeof(*array));
```



free(...)

$$\text{free}\left(\boxed{}\right)$$

addr of beginning of the new block

- Call only once per block
- Call from within scope where malloc(...) was calle for that block

Why not malloc(n * sizeof(int))?

Suppose you start with this:

```
int* a=malloc(n*sizeof(int));
```

.... but then later decide to change int to long long.

```
long long* a =malloc(n *sizeof(int));
!!!                            !!!
```

Forgetting to change sizeof(int) to sizeof(long long) would likely result in a buffer overflow (BAD).

with sizeof(*array) the compiler prevents that.

Analogy

a=malloc($\boxed{n}$)	reserve a hotel room for n people
free($\boxed{a}$)	check out of hotel
$\boxed{a}$	room key

Stack

addr	type*	name*	value	part	fn
200	int	argc	1	args	main(...)
204	char**	argv	→ {"./foo"}	args	
212	void*			ret addr	
220	int	n	2	locals	
224	int*	array	[400]		
232					

Heap

addr	value		🔒
400	10	11	~~🔒~~
408			

Data segment

addr	type*	value
600		

- Type and name are not actually stored in memory or executable. Addresses shown are fictional.
- Assume `sizeof(int)==4` `sizeof(char)==1` `sizeof(void*)==8`
- To show struct types with fields, split the type and name fields. In value field, just write the value of the field. Example →

type	name	value
Point : int,	*p . x*	*5*
: int	*. y*	*6*

Analogy (con't'd)

Beware:

forgetting to call free	→	forgetting to check out of hotel → expensive
calling free but continuing to use the memory	→	checking out but going back into room
calling free 2x	→	checking out twice WT[redacted]
calling free on a different addr	→	checking out of a diff room WT[redacted]