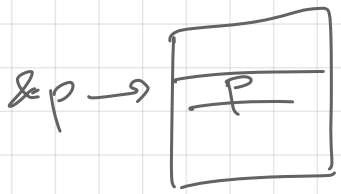
Week 1



Question 9.

```
int x;    // variable at address 1000 with initial value 0.
int *p;   // variable at address 2000 with initial value 0.
```

	statement	x value	p value	x address	p address
initial.	-	0	0	1000	2000
a.	p = &x;	0	1000	1000	2000
b.	x = 5;	5	1000	1000	2000
c.	*p = 3;	3	1000	1000	2000
d	x = (int) p;	1000	1000	1000	2000
e.	x = (int) &p;	2000	1000	1000	2000
f.	p = NULL;	2000	NULL	1000	2000
g.	*p = 1;		fails	because	p is NULL.

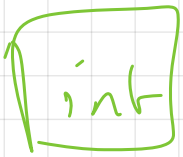
Question 6.

when to use * and/or malloc for structs?

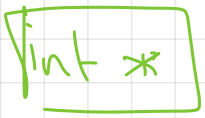
struct node a; use .
- declared on the stack
- don't need malloc.
- only use within a function.

struct node *b; use ->
- declared on the heap
- need malloc
- good for passing between functions

a pointer is a variable that points to the address of a variable.



var1;



var

struct

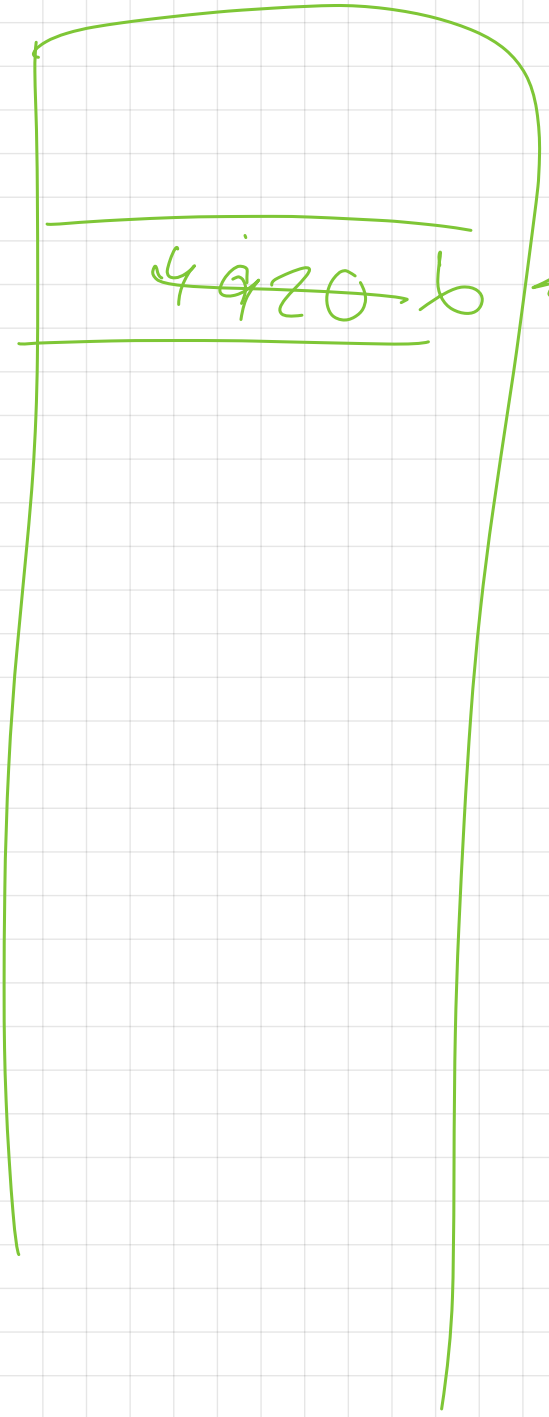
what does sizeof do?

sizeof gets the # of bytes (of variable eg "a" or type eg "int")

what does malloc do?

allocates memory of given size @ the address on LHS.

c = malloc (8) allocates 8 bytes @ c's address
- c: variable to allocate for
- 8: # of bytes



$f(5) = f(4) + f(3)$

$f(3)$ $f(2)$ $f(2)$ $f(1)$

r(0)
↓
r(1)
↓
r(2)
↓
3
4
⋮
10

Week 2

1. When should the types in **stdint.h** be used?

   => what is the type **uint8_t**? (unsigned, 8 bits)
   00000000
   how is it different to **int8_t**? (positive & negative)
   → not unsigned

2. How are the bases [decimal (base 10), hexadecimal (base 16), octal (base 8) and binary (base 2)] denoted in C?

   hexadecimal: starts with 0x
   octal: starts with 0
   decimal: everything else
   binary: => 0b → not standard → pls don't use it.

	decimal	binary (=8 digits)	octal	hexadecimal
a.	1	0 0 0 0 0 0 0 1	0 0 1	0 1
b.	8	0 0 0 0 1 0 0 0 (2^3)	0 1 0	0 8
c.	10	0 0 0 0 1 0 1 0 (2^3+2^1)	0 1 2	0 A
d.	15	0 0 0 0 1 1 1 1	0 1 7	0 F
e.	16			
f.	100			
g.	127			
h.	200			

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

convert 8 to binary

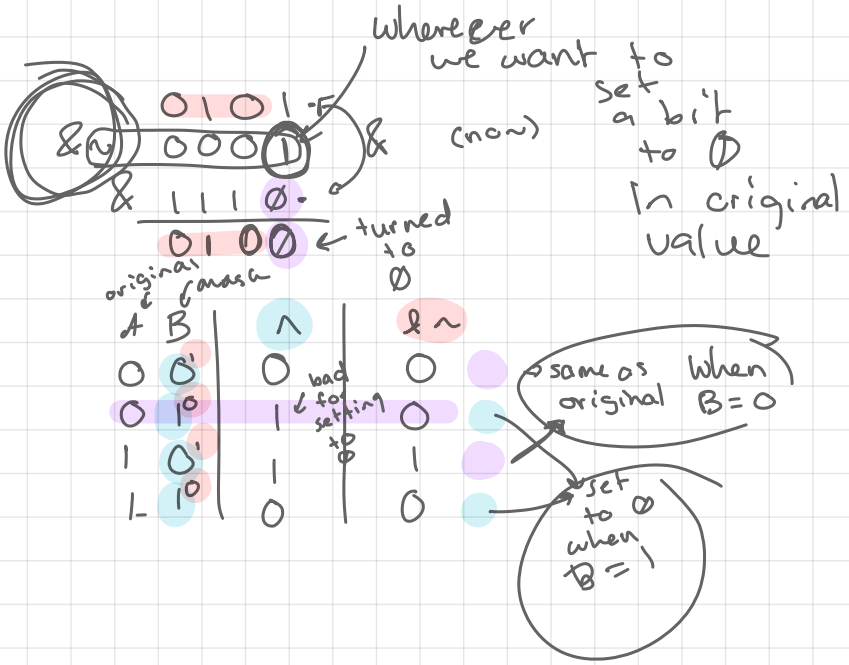
(÷2)	8		
	4	r 0	↑ binary digit
	2	r 0	
	1	r 0	
	0	r 1	

$\Rightarrow 8_{10} = 1000_2$

8_{10}: base 10 (decimal); 1000_2: base 2 (binary)

(÷2)	10		
	5	r 0	↑ 1010_2
	2	r 1	
	1	r 0	
	0	r 1	

(÷2)	15		
	7	r 1	
	3	r 1	
	1	r 1	
	0	r 1	



& 01010 / 11100 / 01000

3. Bitwise Operations

A	B	OR (A | B)	AND (A & B)	XOR (^)
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

A	NOT (~)
0	1
1	0

not as ! same

~ → this is bitwise

a) 0x5555 | 0xAAAA = 0xFFFF

0101 0101 0101 0101
| 1010 1010 1010 1010
1111 1111 1111 1111

b) 0x5555 & 0xAAAA = 0x0000

0101 0101 0101 0101
& 1010 1010 1010 1010
0000 0000

c) 0x5555 ^ 0xAAAA = 0xFFFF

0101 0101 0101 0101
^ 1010 1010 1010 1010
1111 1111

d) 0x5555 & ~0xAAAA

0101 0101 0101 0101 = 0x5555
0101 0101 0101 0101
0101 0101

e) 0x0001 << 6

0000 0000 0000 0001
0000 0000 0100 0000

f) 0x5555 >> 4

0101 0101 0101 0101
0000 0101 0101 0101

g) 0x5555 & (0xAAAA << 1)

1010 1010 1010 1010 << 1
= 0101 0101 0101 0100
=> 0101 0101 0101 0101
& 0101 0101 0101 0100
0101 0101 0101 0100 = 0x5554

h) 0xAAAA | 0x0001

1010 1010 1010 1010
| 0000 0000 0000 0001
1010 1010 1010 1011

i) 0xAAAA & ~0x0001

1010 1010 1010 1010
& 1111 1111 1111 1110
1010 1010 1010 1110

Given a variable X...

Copy/extract the value of a specific bit: X & mask

Set a specific bit to 1: X | mask

Set a specific bit to 0: X & ~mask.

to set a specific bit(s) to 1.

original value => 0001 0101

0011 0000 => mask

try and &

0 → not what we want

try or

1 0011 0101

still same

use to set digits to 1.

set to 1 like we wanted

try copying specific bits from a value:

0101 0101 => original

don't copy → 0110 0000 => mask

& 0100 0000

↑ same as the original

set specific bits to 0.

0101 1010 => original

1001 1111 => mask

& 0001 1010

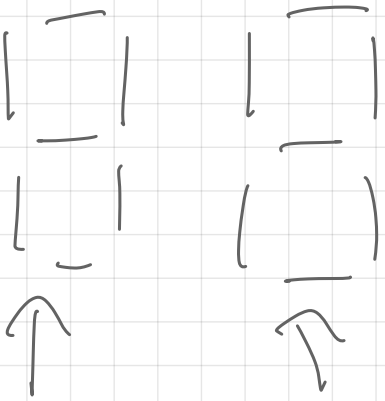
0101 1010 => original

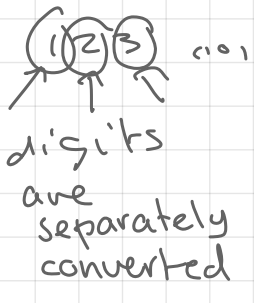
0110 0000 => mask

~&(~mask = 1001 1111)

& 0001 1010

BCD





1_{10} : 1_2 => 0000 0001$_2$

2_{10} : 10_2 => 0000 0010$_2$

3_{10} : 11_2 => 0000 0011$_2$

(8 bits => uncompressed BCD

| 0000 0001 | 0000 0010 | 0000 0011 |

vs normal binary --- 00 0111 1011

compressed BCD

=> 4 bits instead of 8.

258 = | 0000 0001 | 0000 0010 |

1 2