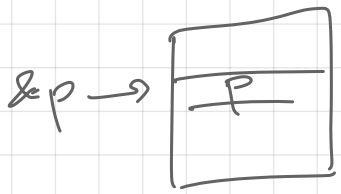
Week 1



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.

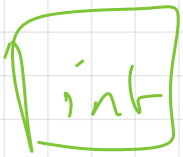
Question 6.

when to use * and/or malloc for structs?

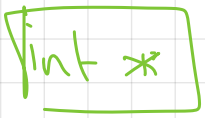
struct node a; use .
- declared on the stack
- don't need malloc.
- only use within a function.

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

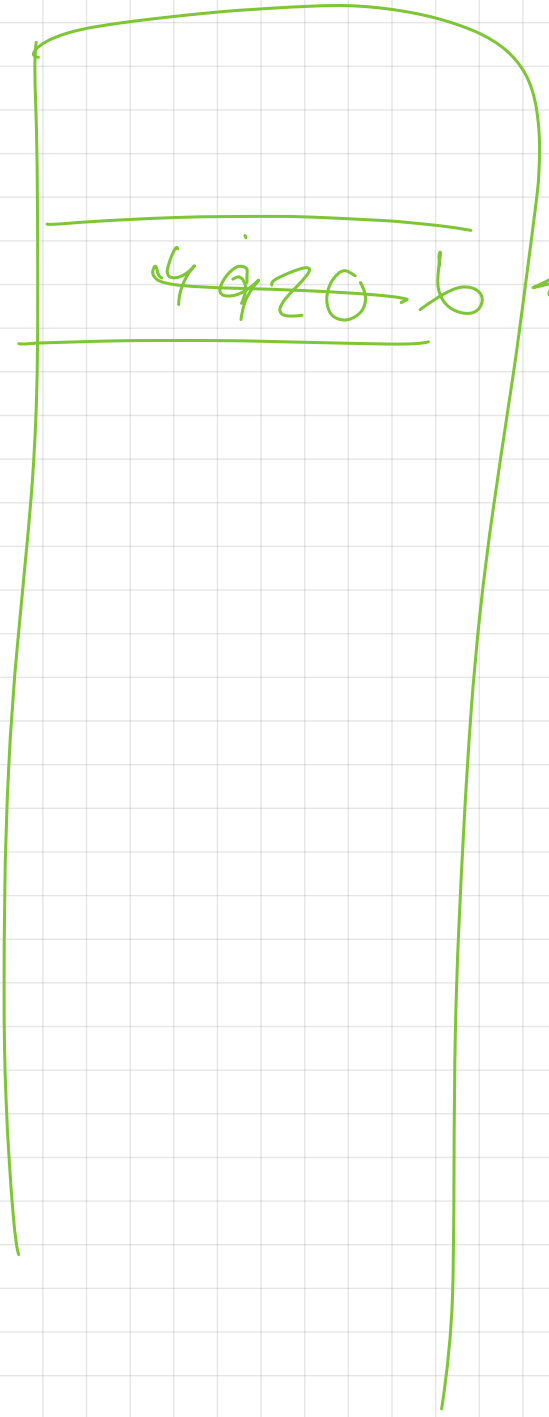
what does sizeof do?

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

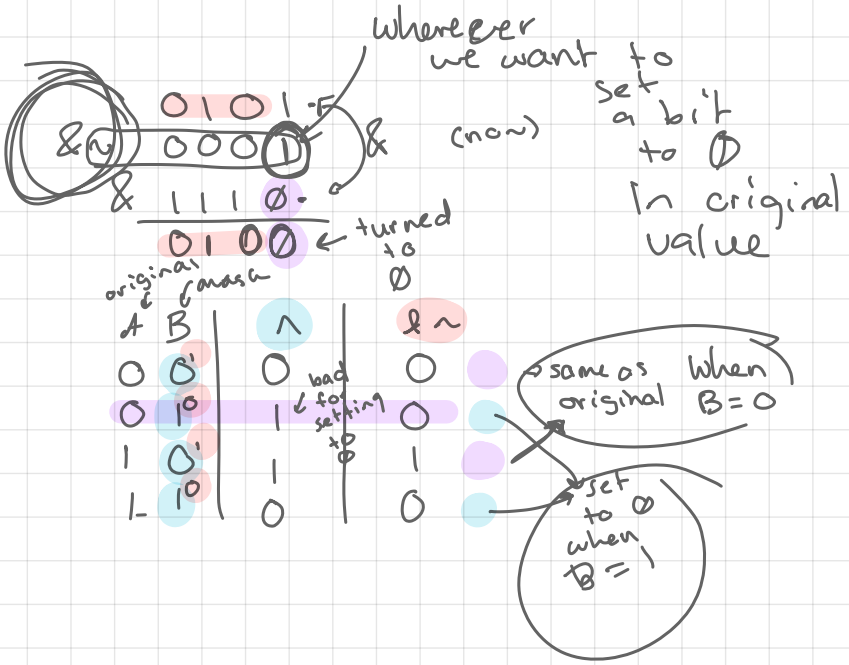
(÷2)	8		
	4	r 0	↑ binary digit
	2	r 0	
	1	r 0	
	0	r 1	

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

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

True = 1 T
False = 0 F

! 101010 → True = 00000
~ 101010 = 010101

not same as !

A | B A & B

A	B	OR |	AND &	XOR ^
→ 0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

A	NOT ~ → this is bitwise
0	1
1	0

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

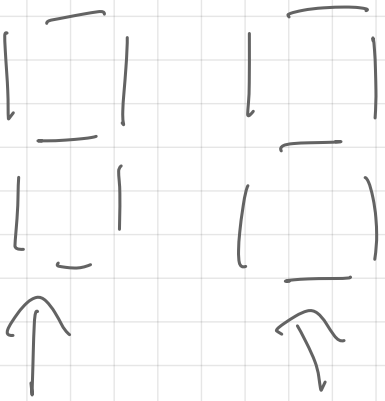
0101 1010 => original

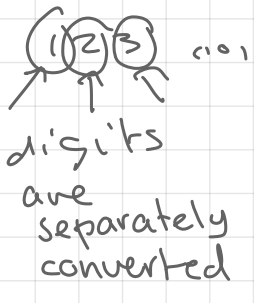
0110 0000 => mask

~&(~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

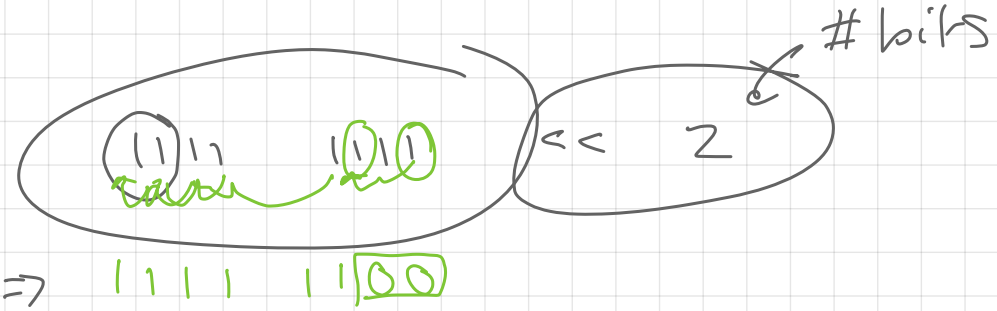
=> 4 bits instead of 8.

258 = | 0000 0001 | 0000 0010 |

1 2

uint8_t a = 5_{10} (= 0000 0101$_2$)
uint8_t b = 10_{10} (= 0000 1010$_2$)

```
  0000 0101
  0000 1010
-----------
(OR)  |  0000 1111
(and) &  0000 0000
(XOR) ^  0000 1111
```

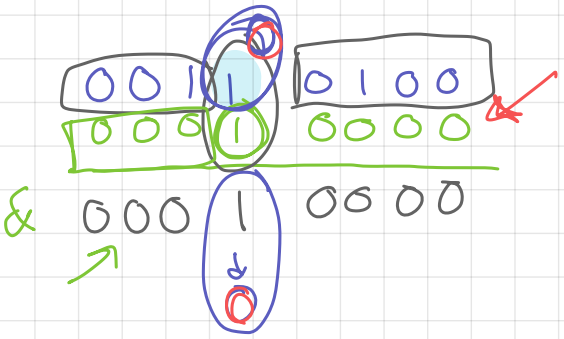


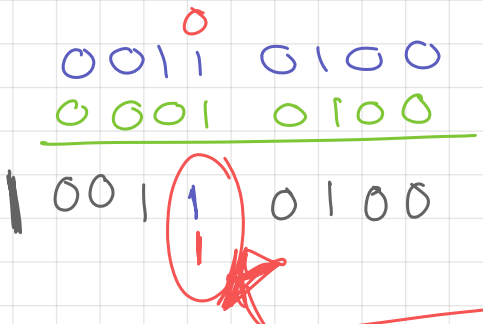
=> 1111 1100

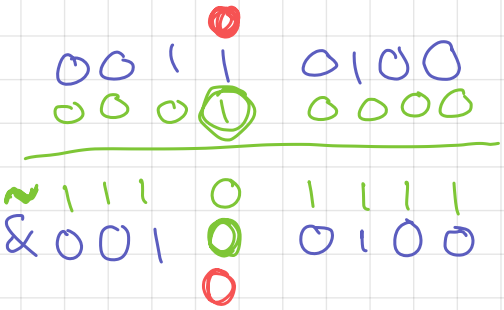
>> 2

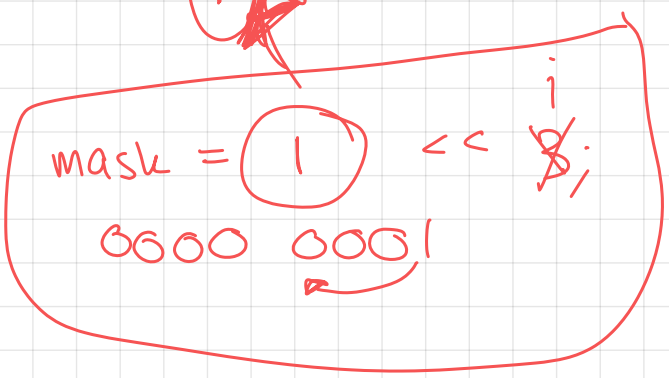
00 11 1111

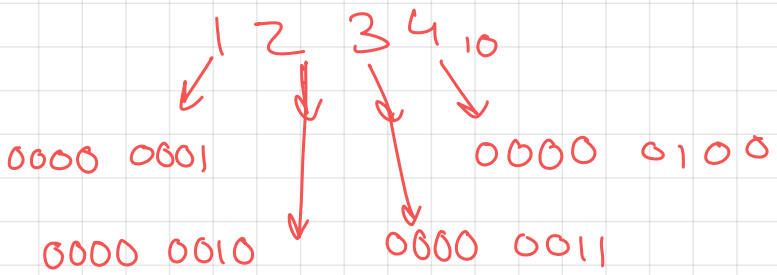
uint .... (unsigned int) ...











1234

0000 0001 | 0000 0010 | 0000 0011 | 0000 0100

258's 0000 0001 | 0000 0010

$1 \times 10^1 + 2 \times 10^0$

Week 3

!! weekly tests start this week !!

Negative Values

How are signed values represented?

X XXX XXXX XXXX XXXX

1 = negative, 0 = positive — if -ve represent using

representing a number with **2's complement**

- for **positive** values:
  use normal bin representation
  signed bit is 0
- for **negative** values:
  set signed bit to 1
  use 2's complement.

to determine a number's representation in 2's complement:

- get normal bin. representation
- invert all bits
- add 1

Eg. 5_{10}:

0000 0000 0000 0101

vs -5_{10}:

invert: 1111 1111 1111 1010

add 1: 1111 1111 1111 1011

$11_{10} = 1011_2$

2's comp = 0100 + 1

$-11_{10} = 0101$

$10_{10} = 1010$

2's c = 0110

100_{10}: invert, keep

0000 0000 0110 0100

← find 1st 1

$10_{10} = 1010$

$= 0101$

$+ \quad 1$

$0110 = -10_{10}$

cheaty method

$11_{10} = 1011_2$

invert everything to left of first 1

$= 0101$

vs -100_{10}:

cheat 1111 1111 1001 1100

invert 1111 1111 1001 0011

+1 1111 1111 1001 1100

convert 16-bit representation to the corresponding decimal value:

- 0x0013 = 0000 0000 0001 0011 $= +19$
  = Positive
  2^4 $2^1 2^0$
- 0xffff = 1111 1111 1111 1111 $= -1$
  = negative
  ∴ need to use 2's complement
  = 0000 0000 0000 0001 = 1

2's complement undoes itself

ie: 2's c (2's c (a)) = a

ie -100 = 1111 1111 1001 1100

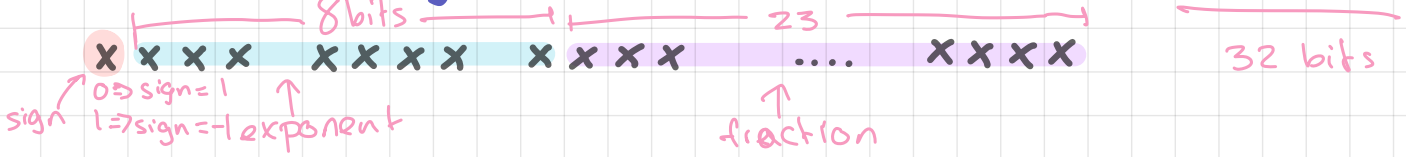
0000 0000 0110 0100

normal method 0000 0000 0110 0011

0000 0000 0110 0100

Floating Point Values

How are floating point values represented? (IEE 754)



from this we can calculate the value with the formula:

$$\text{sign} \times (1 + \text{frac}) \times 2^{\text{exp} - 127}$$

note: we don't use two's complement here.

Eg: Convert the following to decimal numbers

10^{-1} 10^{-2} 10^{-3} 10^{-4}
0. 1 2 3 4
$0.1 = 1 \times 10^{-1}$
$0.02 = 2 \times 10^{-2}$
∴ . . .

f) 0 | 1 0 0 0 0 0 0 0 | 0 1 1 0 ... 0
(7 6 5 4 3 2 1 0 | -1 -2 -3 -4 ... -23)

sign: positive ∴ = 1
exp: $2^7 = 128$
frac: $2^{-2} + 2^{-3} = 0.375$
value = $1 \times (1 + 0.375) \times 2^{128 - 127} = 2.75$

c) 0 | 0 1 1 1 1 1 1 1 | 1 0 0 ... 0
(2^7 2^6 ... 2^1)

sign: = 1 (+ve)
exp: = 127
frac: $= 2^{-1} = 0.5$
value = $1 \times (1 + 0.5) \times 2^{127 - 127} = 1.5$
(1 + 0.5 = 1.5; $2^0 = 1$)

don't follow the formula
denormal
$X \cdot Y$ where $X = 0$

a) 0 | 0 0 0 0 0 0 0 0 | 0 0 0 ... 0

sign: 1
exp: 0
frac: 0
value = $1 \times (1 + 0) \times 2^{0 - 127} = 2^{-127} \simeq 0$
(1×2^{-127})

b) 1 | 0 0 0 0 0 0 0 0 | 0 0 0 ... 0

sign: -1
exp: 0
frac: 0
value = $\_\_ \times (1 + \_\_\_\_) \times 2^{\_\_ - 127} = -2^{-127} = -0$

255 = 1111 1111

0 ——— want 0 here ——— 255

Eg 2: convert the following to IEEE 754 encoded bit strings:

a) $2.5 \div 2^1 = 1.25 \Rightarrow 2.5 = 1.25 \times 2^1$ ($\times 2$, $\times 2$)

$(1 + 0.25) \times 2^1$

sign: +ve ∴ 0

exp: exp − 127 = 1 ⇒ exp = 1 + 127 = 128

frac: 010....0

bits = 0 1000 0000 010....0

> first express the number in the form: $(1 + frac) \times 2^n$

b) $0.375 \times 2^2 = 1.5 \Rightarrow 0.375 = \frac{(1+0.5)}{2^2} \times 2^2 = (1+0.5) \times 2^{-2}$

sign: 0

exp: exp − 127 = −2 ⇒ exp = −2 + 127 = $125_{10} = 0111\ 1101_2$

frac: 100....0

bits = 0 0111 1101 100....0

how to convert fract. to bin?

(×2) 0.25 → 0.5 → 1.0

c) $27.0 \div 2^4 = 1.6875 \Rightarrow 27.0 = (1+0.6875) \times 2^4$

sign: 0

exp: 4 = exp − 127 = 13_{10} = 1000 0011$_2$

frac: 0.6875_{10} = 1011 00....0

bits = 0 1000 0011 1011 0....0

0.6875 → 1.375 (×2) → 0.75 → 1.5 (×2) → 1.0

steps:
- start w/ fraction component.
- multiply by 2 until either we get 0 or run out of space for digits
- take the digit on the LHS as the next bin. digit.
- take the value on RHS and repeat process

d) 100

sign:

exp:

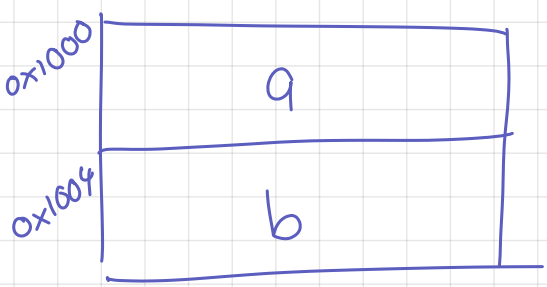
frac:

bits =

Structs vs Unions

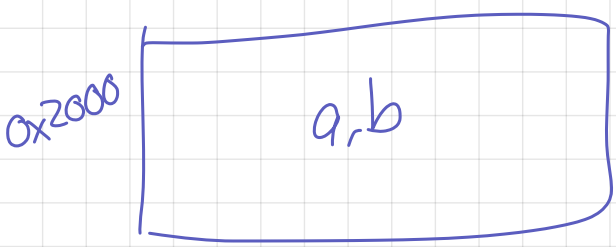
```
struct {
    int a;      // assume 4 bytes
    float b;
} x1;
```

&x1 = 0x1000
&(x1.a) = 0x1000
&(x1.b) = 0x1004




```
union {
    int a;
    float b;
} x2;
```

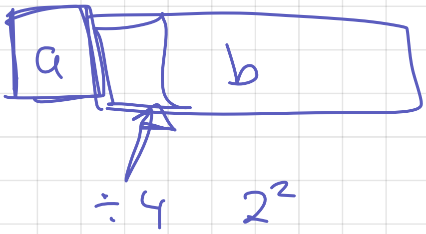
&x2 = 0x2000
&(x2.a) = 0x2000
&(x2.b) = 0x2000

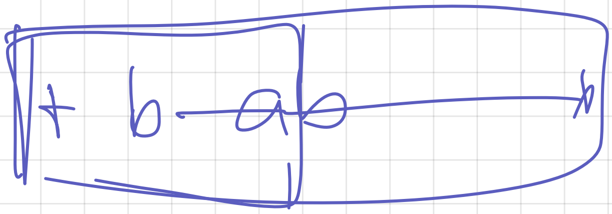


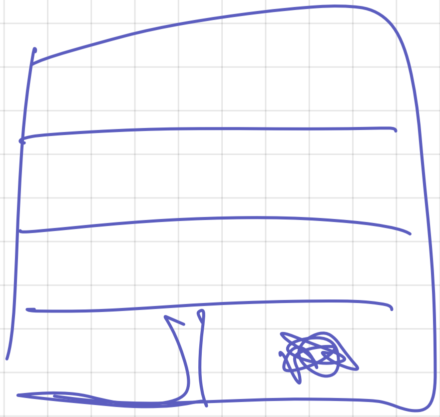
x2.a = 10

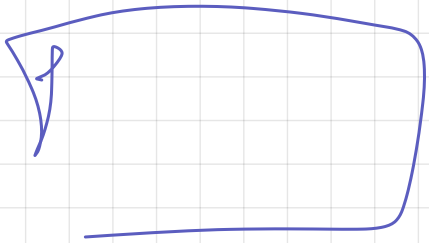
x2.b = $1(1 + 2^{-20} + 2^{-22}) \times 2^{0-127}$

= $1(1.\quad\quad) 2^{-127}$

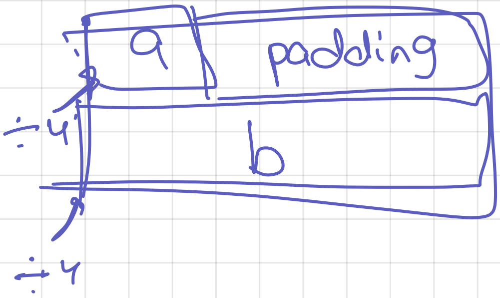








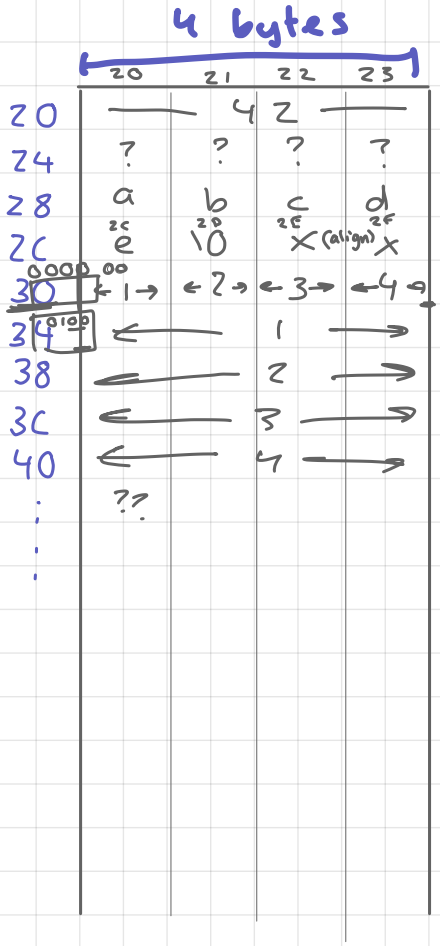
what actually happens

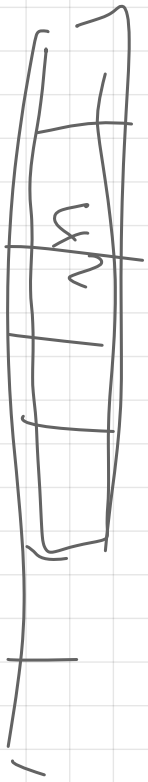


$4 + 4 = 8$

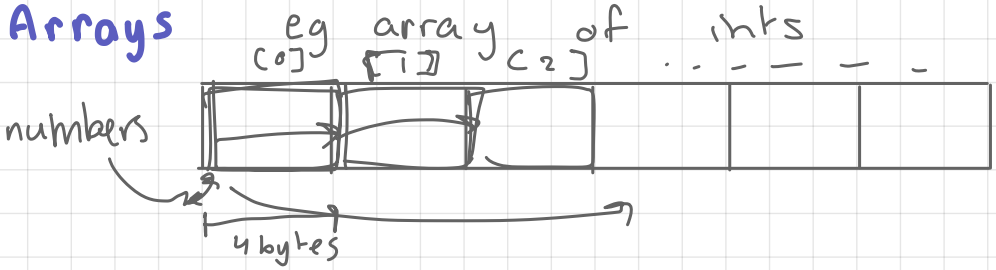
Week 5

3. 0x10000020





Arrays



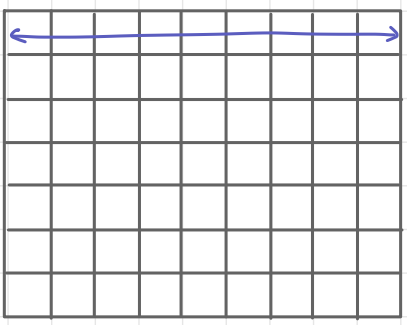
calculate index address:

(1d) address = starting address of array + offset

offset = index x size of elements

1d array_address = label + index x sizeof(element)

2d array_address = label + row x NUMCOLS x sizeof(element) + index x sizeof(element)



(week 7 content) from wk 5 !!

Functions

to "call another fn"
→ use jal address
↳ changes $ra to address of next instr...

main
caller
calls another function

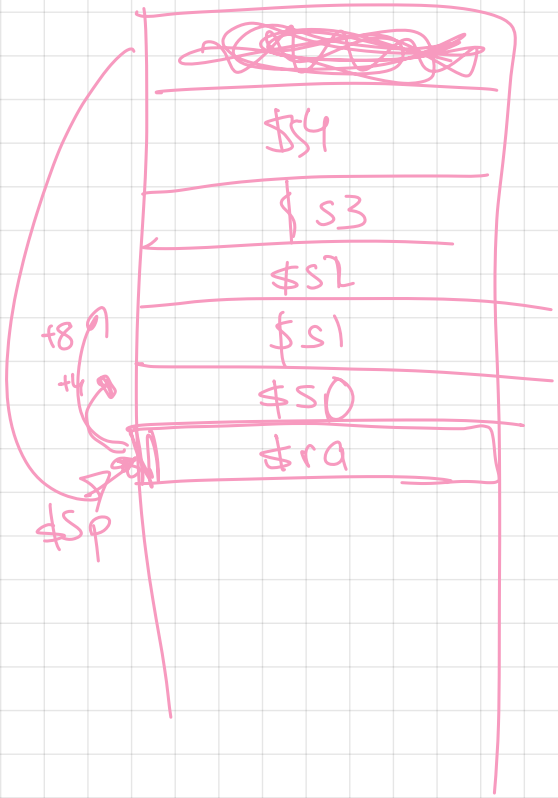
can assume is $s aren't changed by callee.

main
callee
is being called

can assume $t registers are free to use (don't need to save them)

have to save the $S registers if using them.
→ same for $ra

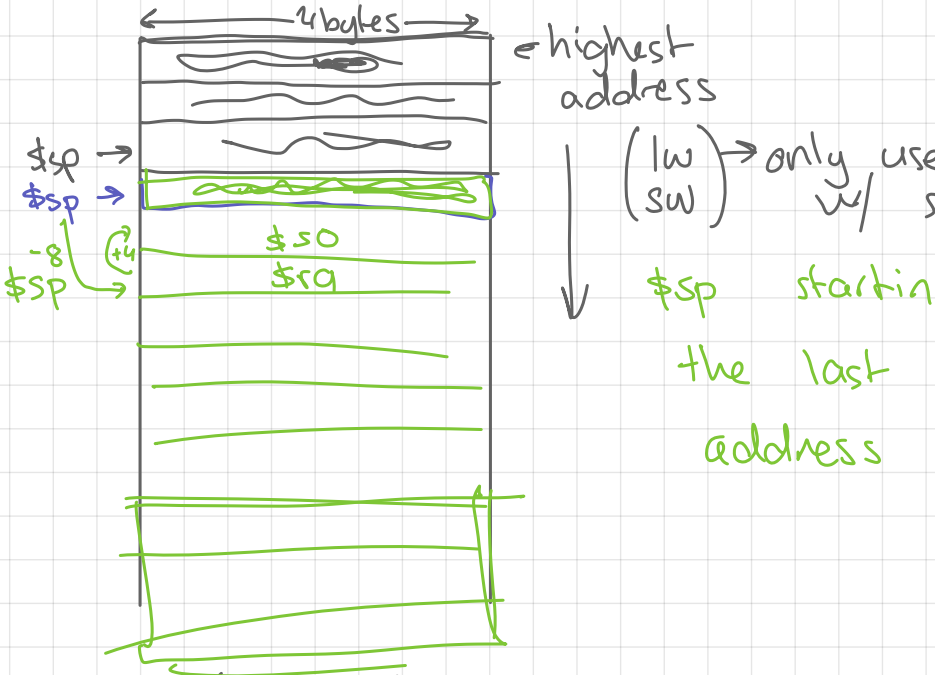
to give arguments use $a registers in order!

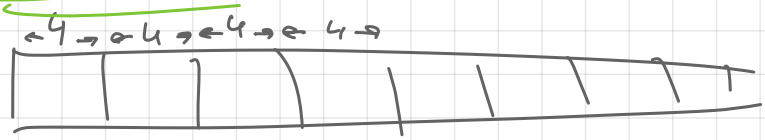


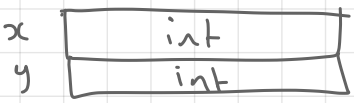
Actual Week 7

	Rights	Responsibilities
callee function being called	free to use: (change them as much as want) • $t • $a arguments: in $a regs (extras go on stack)	needs to save/restore: • $s regs • $ra • $fp return value in: $v0 (and v1 if 4 < value ≤ 8 bytes)
caller function doing the calling	$ra, $s aren't changed by callee	put arguments in $a regs if it really wants to use $t before & after (w/ same value) has to save it itself.

The stack

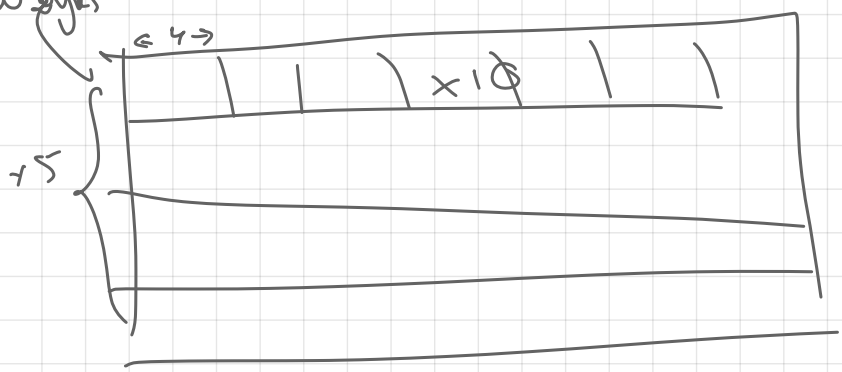


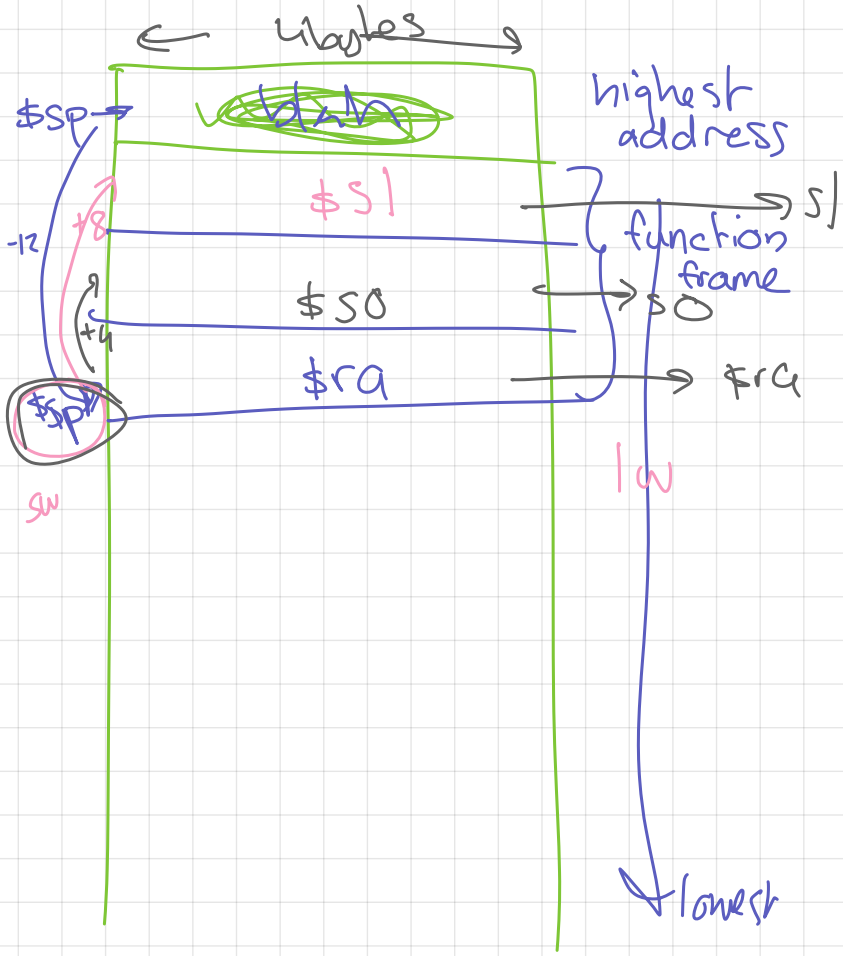


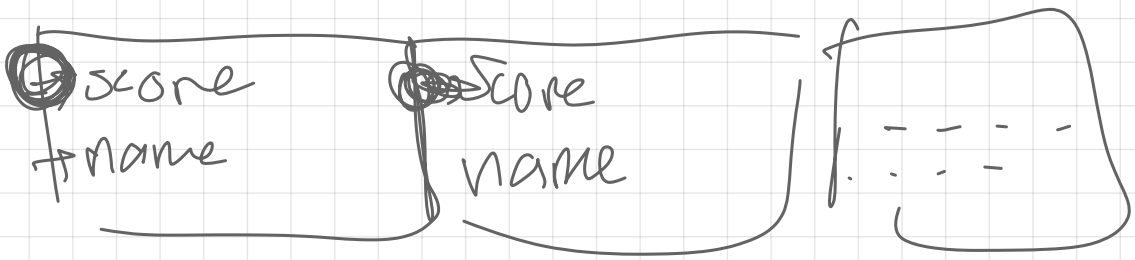


4x10x5
=40x5
=200 bytes

4 * 20 = 80 bytes
bytes elements

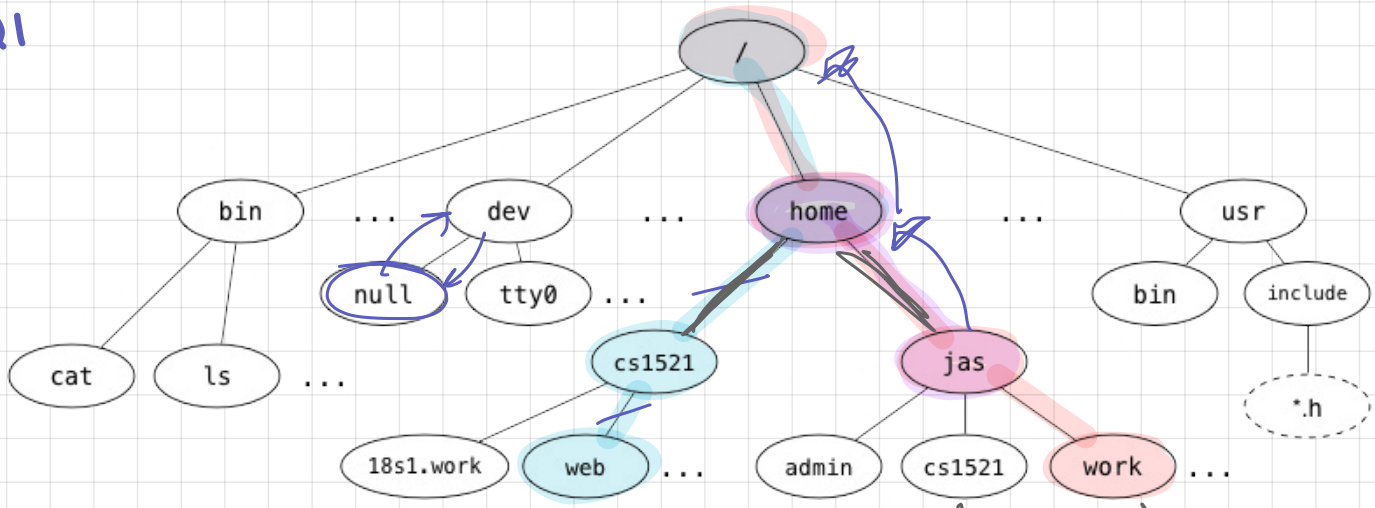



4bytes
$sp
highest address
$s1
-12
+8
function frame
$s0
+4
$ra
$sp
sw
lw
lowest


score
name
score
name

Week 8

Q1



a) /home/cs1521/web
/home/jas/work

causes (h) a cycle

b) ~ = home
.. = parent directory

home/jas/../../ = /

d) cat? → executable file.

e) home → directory

f) tty0 → files that act as output to peripherals

g) symbolic link is a pointer to another file in the tree.

h) graph structure has cycles

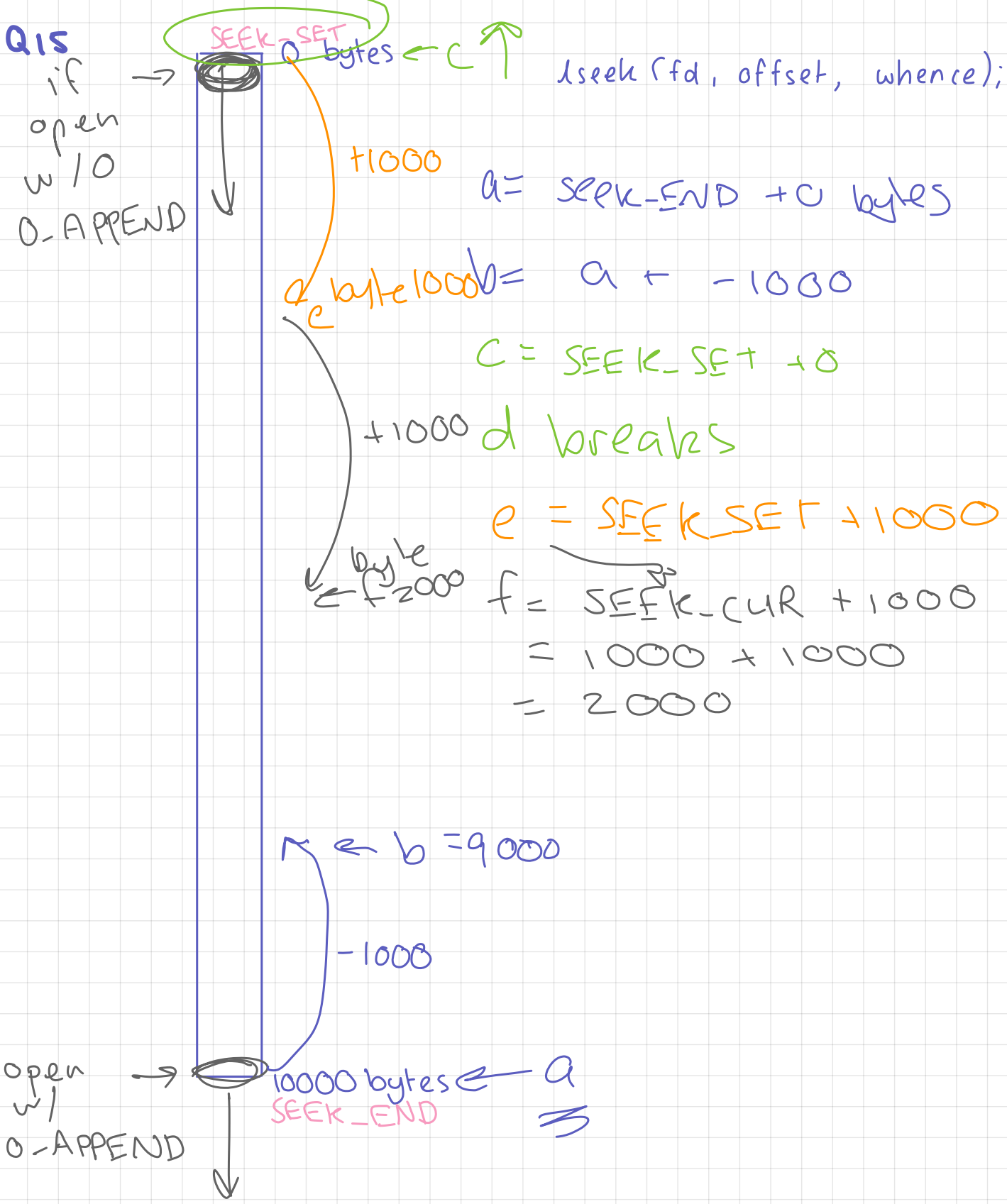
→ lookup hard links!

Q3 try `man 3 fopen` in your terminal!

Q13

```
fopen (pth, "r");
    open ( pth,  O_RDONLY);
fopen ( pth , "a");
    open (pth,  O_WRONLY | O_CREAT | O_APPEND);
fopen (pth, "w");
    open (pth,  O_WRONLY | O_CREAT | O_TRUNC);
fopen(pth , "r+");
    open (pth,  O_RDWR)
```

Q15


SEEK_SET
0 bytes
c
lseek(fd, offset, whence);
if open w/ O_APPEND
+1000
a = SEEK_END +0 bytes
a byte 10000 b = a + -1000
c = SEEK_SET +0
+1000
d breaks
e = SEEK_SET +1000
byte 2000 f
f = SEEK_CUR +1000
= 1000 + 1000
= 2000
b = 9000
-1000
open w/ O_APPEND
10000 bytes
SEEK_END
a
3