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

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

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

int p *var;
int *
p var
int
var;

struct

what does sizeof do?

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

what does malloc do?

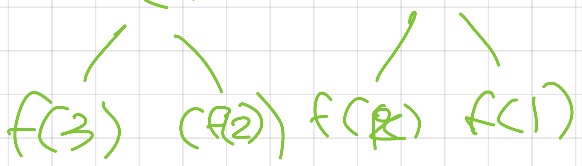
allocates memory of given size @ the address on LHS.

c = malloc (8) allocates 8 bytes @ c's address

↑ variable to allocate for

↑ ~~#~~ of bytes

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



r(0)

↓

r(1)

↓

r(2)

↓

3

4

⋮

↓

10

Week 2

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

unsigned

⇒ what is the type uint8_t ?

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.

=8 digits

	decimal	binary	octal	hexadecimal
a.	1	0 0 0 0 0 0 0 1	0 0 1	0 1
b.	8	2 1 0 2 1 0 → 0 0 0 | 0 0 1 | 0 0 0 ↑ oct: $\cancel{2^2}+\cancel{2^1}+2^0$ $\cancel{2^2}+\cancel{2^1}+\cancel{2^0}$ hex | 0 0 0 0 | 1 0 0 0 | 0 $2^3+\cancel{2^2}\cancel{2^1}\cancel{2^0}=8$	0 1 0	0 8
c.	10	0 0 0 0 0 | 1 0 1 0 $2^3\cancel{2^2}$ 2^1 $\cancel{2^0}$	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
2 r 0
1 r 0
0 r 1

↑ binary digit

⇒ $8_{10} = 1000_2$

↑ base 10 (decimal)

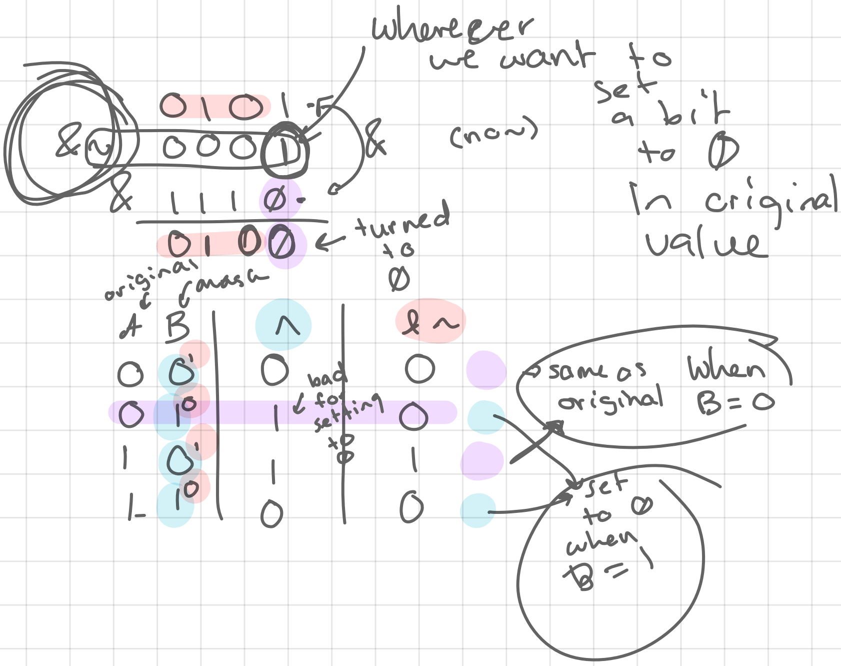
↑ base 2 (binary)

(÷2) 10
5 r 0
2 r 1
1 r 0
0 r 1
↑

1010_2

(÷2) 15
7 r 1
3 r 1
1 r 1
0 r 1

```
  01010
& 11100
  01000
```



0101
&~ 0001
& 1110
0100 ← turned to Ø

& (no~)

wherever we want to set a bit to Ø in original value

original → A, mask → B

A	B	^	&~
0	0^1	0	0
0	1^0	1	0
1	0^1	1	1
1	1^0	0	0

bad for setting to Ø

same as original when B=0

set to Ø when B=1

3. Bitwise Operations

True = 1 T
False = 0 F

!|101010| → True = 00000
~|101010| = 010101 not as same!

A	B	A | B OR |	A & B AND &	XOR ^	A	NOT ~ → this is bitwise
→ 0	0	0	0	0	0	1
0	1	1	0	1	1	0
1	0	1	0	1		
1	1	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

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

use to set digits to 1.

set to 1 like we wanted

still same

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 ∅.

0101 1010 => original
1001 1111 => mask

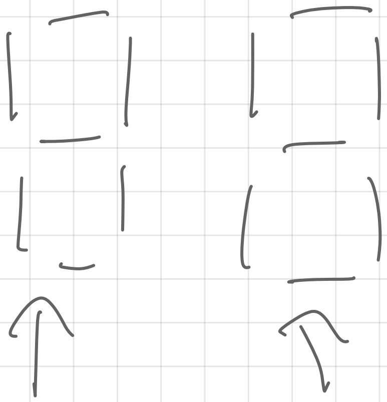
& 0001 1010

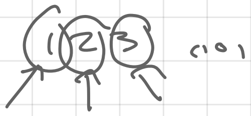
0101 1010 => original
0110 0000 => mask

~& (~mask =
1001 1111)

& 0001 1010

BCD





digits are separately converted to binary

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

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

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

(8bits => uncompressed BCD

$$\boxed{0000\ 0001\ |\ 0000\ 0010\ |\ 0000\ 0011}$$

vs normal binary --- $\boxed{00\ 0111\ 1011}$

compressed BCD

=> 4 bits instead of 8.

$$258 = \left|\underbrace{0000\ 0001}_{1}\right|\underbrace{0000\ 0010}_{2}\right|$$

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 1111 << 2   # bits

⇒ 1111 11$\boxed{00}$

>> 2

$\boxed{00}$11 1111

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

0011 0100
0001 0000

& 0001 0000
0

0011 0100
0001 0100

| 0011 0100
1

mask = 1 << ~~8~~ i;
0000 0001

0
0011 0100
0001 0000

~ 1110 1111
& 0010 0100
0

$1\ 2\ 3\ 4_{10}$ 1234

0000 0001 0000 0100

0000 0010 0000 0011

0000 0001 | 0000 0010 | 0000 0011 | 0000 0100

258's $\underbrace{0000\ 0001}_{1 \times 10^1}\ |\ \underbrace{0000\ 0010}_{+\ 2 \times 10^0}$

Week 3

!! weekly tests start this week !!

Negative Values

How are signed values represented?

x x x x x x x x x x x x x x x x

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. representatio
- 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

100_{10}: invert keep

0000 0000 0110 0100 ← find the first 1

c

vs -100_{10}:

cheat 1111 1111 1001 1100

invert 1111 1111 1001 1011

+1 1111 1111 1001 1100

11_{10} = 1011_2

2's comp = 0100 + 1

-11_{10} = 0101

10_{10} = 1010_2

2'sc = 0110

10_{10} = 1010

= 0101

+ 1

0110 = -10_{10}

cheaty method

11_{10} = 1011_2

first 1

invert everything to left of the 1

= 0101

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'sc (2'sc (a)) = a

ie -100 = 1111 1111 1001 1100

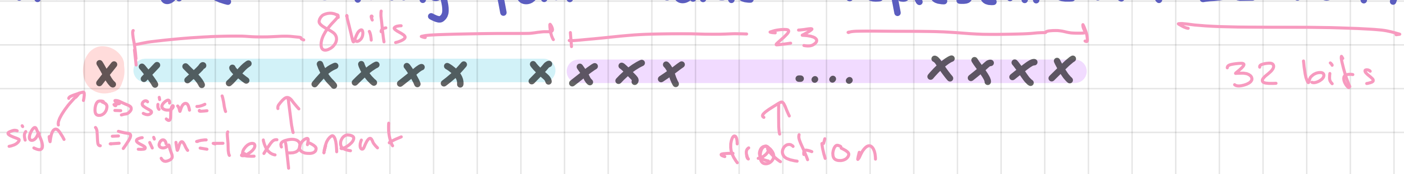
0000 0000 0110 0100

normal 0000 0000 0110 0011

method 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:

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

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

Eg: Convert the following to decimal numbers

$0.\overset{10^{-1}}{1}\overset{10^{-2}}{2}\overset{10^{-3}}{3}\overset{10^{-4}}{4}$

$0.1 = 1\times10^{-1}$

$0.02 = 2\times10^{-2}$

...

f) $\boxed{0}$ $\overset{7}{1}\overset{6}{0}\overset{5}{0}\overset{4}{0}\overset{3}{0}\overset{2}{0}\overset{1}{0}\overset{0}{0}$ | $\overset{-1}{0}\overset{-2}{\underline{1}}\overset{-3}{\underline{1}}\overset{-4}{0}\ldots\overset{-23}{0}$

sign: positive ∴ = 1

exp: $2^7 = 128$

frac: $2^{-2} + 2^{-3} = 0.375$

value = $\underline{1} \times (1 + \underline{0.375}) \times 2^{\underline{128}-127} = 2.75$

c) 0 $0\overset{2^7}{}\overset{2^6}{1}111111$ $\overset{2^{-1}}{1}00\ldots0$

sign: = 1 (+ve)

exp: = 127

frac: $= 2^{-1} = 0.5$

value = $\underline{1} \times (\underbrace{1 + \underline{0.5}}_{=1.5}) \times \underbrace{2^{\underline{127}-127}}_{2^0=1} = 1.5$

a) 0 00000000 000...0

sign: 1

exp: 0

frac: 0

value = $\underline{1} \times (\underbrace{1 + \underline{0}}_{1}) \times 2^{\underline{0}-127} = 2^{-127} \approx \underline{\underline{0}}$

$\times 2^{-127}$

don't follow the formula

'denormal'

X ∘ Y where X=0

b) $\boxed{1}$ 00000000 000...0

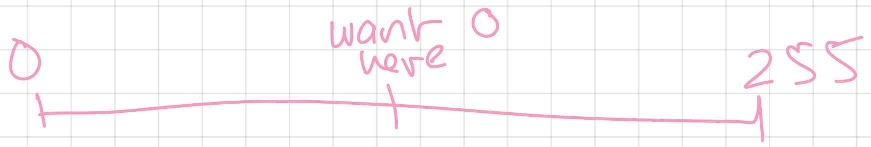
sign: -1

exp: 0

frac: 0

value = $\underline{\quad} \times (1 + \underline{\quad}) \times 2^{\underline{\quad}-127} = -2^{-127} = -0$

255 = 1111 1111



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

a) $2.5 \overset{\times 2}{\div 2} = \overset{\times 2}{1.25} \Rightarrow 2.5 = 1.25 \times 2$

$(1 + \boxed{0.25}) \times 2^{1}$

sign: +ve ∴ 0

exp: $exp - 127 = 1 \Rightarrow exp = 1 + 127 = \underline{\underline{128}}$

frac: 0 1 0 0

bits = 0 1000 0000 0 1 0 0

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

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

sign: 0

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

frac: 100 0

bits = 0 0111 1101 100 0

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

sign: 0

exp: $4 = exp - 127 = 131_{10} = 1000\ 0011_2$

frac: 0.6875_{10} = 1011 00 0

bits = 0 1000 0011 1011 0 0

d) 100

sign:

exp:

frac:

bits =

how to convert fract. to bin?

(×2) 0.25

0.5

1.0

0.5

1.0

0.6875

1.375

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

Structs vs Unions

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

&x1 = 0x1000

&(x1.a) = 0x1000

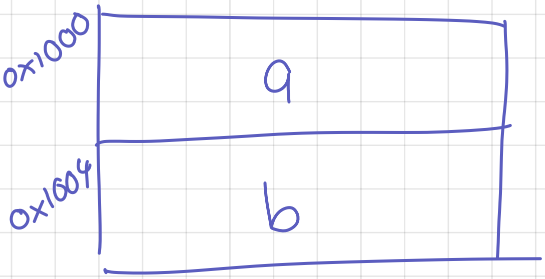
&(x1.b) = 0x 1004

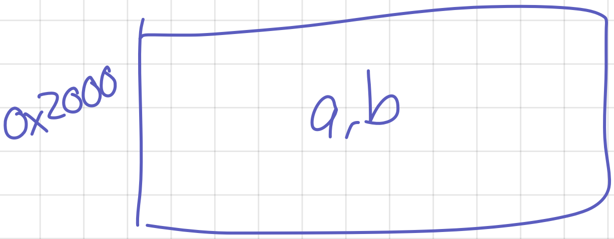
```
union {
    int a;
    float b;
} x2;
```

&x2 = 0x2000

&(x2.a) = 0x2000

&(x2.b) = 0x200

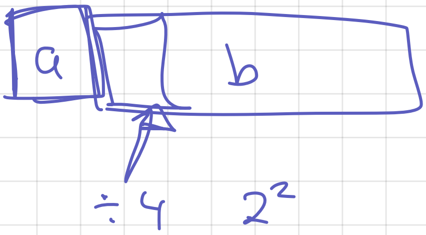


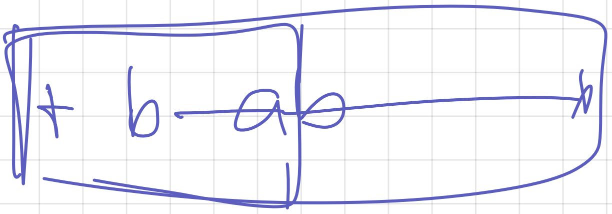


x2.a = 10

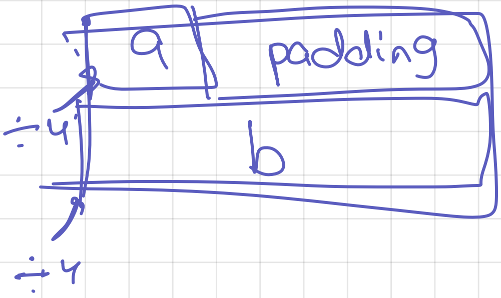
$x2.b = 1(1 + 2^{-20} + 2^{-22}) \times 2^{0-127}$

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





what actually happens



4
$+$
$4 = 8$