

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 (p points to)	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; // typecast	1000	1000	1000	2000
e. x = (int)&p;	2000	1000	1000	2000
f. p = NULL;	? 2000	NULL	1000	2000
g. *p = 1;	fail	because p is NULL		

2000 | P 1000
 1000 | x 5

Question 6.

when to use * and/or malloc for structs?
struct node a;

see code
a.b.

struct node * b;

what does malloc do?

Struct
can hold different
variable types

fields are
accessed by
names

Array
only one
type

fields are
accessed by
indexes

Week 2

int \rightarrow any # of bytes (at least 2 bytes)

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

\Rightarrow what is the type uint8_t?
 unsigned int that is 8 bits

how is it different to int8_t?
 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: any other case
 binary: \rightarrow 0b \rightarrow not standard \rightarrow don't use !!

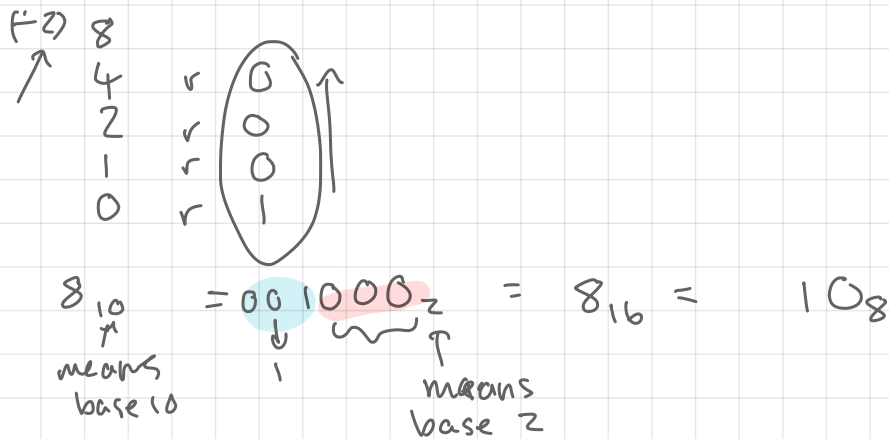
	decimal	binary	octal	hexadecimal
a.	1	0 0 0 0 0 0 0 1	0 0 1	0 1
b.	8	0 0 0 0 1 0 0 0	0 1 0	0 8
c.	10			
d.	15	0 0 0 0 1 1 1 1 $2^0 \quad 2^2 \quad 2^1 \quad 2^0$	0 1 7	0 F
e.	16			
f.	100			
g.	127			
h.	200			

15
 7 r 1
 3 r 1
 1 r 1
 0 r 1

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

Decimal	Hexadecimal	Binary
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

decimal \rightarrow binary (by hand)



binary \rightarrow hex

4 digits of binary are equal to 1 digit in hex.
 \uparrow
 hexadecimal

$\begin{matrix} 1 & 1 & 1 & 1 \\ 2^3 & 2^2 & 2^1 & 2^0 \end{matrix}$

 $15 = F$

 \uparrow

 $\begin{matrix} 1 & 0 & 1 & 1 \\ 2^3 & 2^2 & 2^1 & 2^0 \end{matrix}$

 $2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 1$

(bin \rightarrow dec)

binary \rightarrow octal

same as bin \rightarrow hex but 3 digits

$111_2 = 7$

3. Bitwise Operations

	A	B	OR	AND &	XOR ^		A	NOT ~
1.	0	0	0	0	0		0	1
2.	0	1	1	0	1		1	0
3.	1	0	1	0	1			
4.	1	1	1	1	0			
	True = 1 False = 0		at least one is 1	need both to be 1	only one can be 1			invert the input

a) $0x5555 \mid 0xAAAA$

$$\begin{array}{cccc} 0101 & 0101 & 0101 & 0101 \\ 1010 & 1010 & 1010 & 1010 \end{array} \quad \text{0x5555}$$

$$\begin{array}{cccc} 1010 & 1010 & 1010 & 1010 \\ 0101 & 0101 & 0101 & 0101 \end{array} \quad \text{0xAAAA}$$

$$\hline \begin{array}{cccc} 1111 & 1111 & 1111 & 1111 \end{array} \quad \text{0xFFFF}$$

b) $0x5555 \& 0xAAAA$

$$\begin{array}{cccc} 0101 & 0101 & 0101 & 0101 \\ 1010 & 1010 & 1010 & 1010 \end{array} \quad \text{0x5555}$$

$$\begin{array}{cccc} 1010 & 1010 & 1010 & 1010 \\ 0101 & 0101 & 0101 & 0101 \end{array} \quad \text{0xAAAA}$$

$$\hline \begin{array}{cccc} 0000 & 0000 & 0000 & 0000 \end{array} \quad \text{0x0000}$$

c) $0x5555 \wedge 0xAAAA$

$$\begin{array}{cccc} 0101 & 0101 & 0101 & 0101 \\ 1010 & 1010 & 1010 & 1010 \end{array} \quad \text{0x5555}$$

$$\begin{array}{cccc} 1010 & 1010 & 1010 & 1010 \\ 0101 & 0101 & 0101 & 0101 \end{array} \quad \text{0xAAAA}$$

$$\hline \begin{array}{cccc} 1111 & 1111 & 1111 & 1111 \end{array} \quad \text{0xFFFF}$$

d) $0x5555 \& \sim 0xAAAA$

$$\begin{array}{cccc} 0101 & 0101 & 0101 & 0101 \\ 1010 & 1010 & 1010 & 1010 \end{array} \quad \text{0x5555}$$

$$\begin{array}{cccc} 0101 & 0101 & 0101 & 0101 \\ 1010 & 1010 & 1010 & 1010 \end{array} \quad \text{0xAAAA}$$

$$\hline \begin{array}{cccc} 0101 & 0101 & 0101 & 0101 \end{array} \quad \text{0x5555}$$

e) $0x0001 \ll 6$

$$\begin{array}{cccc} 0000 & 0000 & 0000 & 0001 \\ 00 & 00 & 00 & 00 \end{array} \quad \text{0x0001}$$

$$\hline \begin{array}{cccc} 0000 & 0000 & 0000 & 0000 \end{array} \quad \text{0x0000}$$

f) $0x5555 \gg 4$

always use unsigned vars!!

$$\begin{array}{cccc} 0101 & 0101 & 0101 & 0101 \\ 0000 & 0101 & 0101 & 0101 \end{array} \quad \text{0x5555}$$

$$\hline \begin{array}{cccc} 0000 & 0101 & 0101 & 0101 \end{array} \quad \text{0x0000}$$

g) $0x5555 \& (0xAAAA \ll 1)$

h) $0xAAAA \mid 0x0001$

i) $0xAAAA \& \sim 0x0001$

$$\begin{array}{c} 1000 \\ 2^3 \end{array}$$

Given a variable X ...

Copy / extract the value of a specific bit: $X \& \text{mask}$

Set a specific bit to 1: $X \mid \text{mask}$

Set a specific bit to 0: $X \& \sim \text{mask}$

invert a specific bit: $X \wedge \text{mask}$

set a specific bit to 1.

mask: $\begin{array}{cc} 0011 & 0000 \\ 0000 & 0100 \end{array}$

want: $\begin{array}{cc} 0011 & 0100 \\ 0011 & 0100 \end{array}$

↑ want to set the original value to 1 @ this digit

↑ good

$\begin{array}{cc} 0011 & 0100 \\ 0000 & 0100 \\ \hline 0011 & 0000 \end{array}$

↑

is $a \wedge b == a \& \sim b$?

a	b	$\sim b$	$a \wedge b$	$a \& \sim b$
0	0	1	0	0
0	1	0	0	0
1	0	1	1	1
1	1	0	0	0

no, they're different (2)

$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$

$00000 = 0$

1000

$i = 31$
 $\text{extract} = 31$
 $\text{setting} = 31 - 31 = 0$

$i = 28$
 $\text{extract} = 28$
 $\text{setting} = 31 - 28 = 3$

set a specific bit to 0.

a orig $\begin{array}{cc} 0110 & 1000 \\ 0010 & 0001 \end{array}$

b mask $\begin{array}{cc} 0110 & 1000 \\ 0010 & 0001 \end{array}$

$a \& \sim b$ $\begin{array}{cc} 0100 & 1000 \\ 1101 & 1110 \end{array}$

$a \& \sim b$ $\begin{array}{cc} 0100 & 1000 \end{array}$

copy specific bit:

orig mask $\begin{array}{cc} 0110 & 1010 \\ 0011 & 0000 \end{array}$

want: $\begin{array}{cc} 0010 & 0000 \end{array}$

try & $\begin{array}{cc} 0010 & 0000 \end{array}$

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

0 = positive

1 = negative

representing a number with 2's complement

* for positive values:

we don't use 2's comp.

same representation (normal binary)

* for negative values:

use 2's complement

for a int8-t variable

Eg. 5_{10} :

0000 0101₂

vs -5_{10} :

Use 2's complement: invert all binary digits and then add 1.

1111 1010 + 1 = 1111 1011
signifying that this is negative.

100_{10} :

0110 0100

vs -100_{10} :

invert 1001 1011

add 1: 1001 1100₂

-28 : $28_{10} = 0001 1100_2$
 $\therefore -28_{10} = 1110 0100_2$
invert find 1st 1 from RHS keep the same

convert 16-bit representation to the corresponding decimal value:

• $0x0013$

0000 0000 0001 0011
+ve
 \therefore read as normal
 $2^4 + 2^1 + 2^0 = 19_{10}$

• $0xffff$ $\Rightarrow f = 1111$

1111 1111 1111 1111
-ve
two's complement.
= -1

in order to get the magnitude original, apply 2's comp. again:

0000 0000 0000 0001 = 1

0x 800f

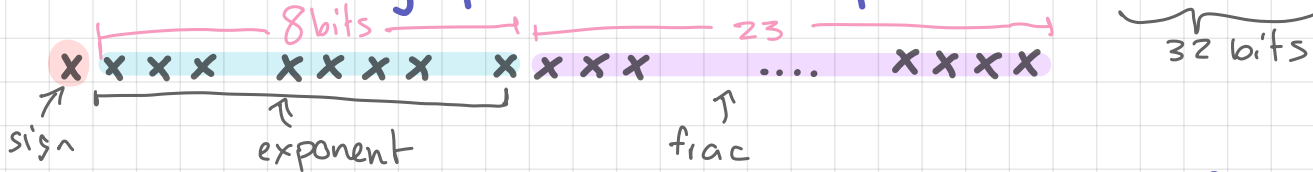
^{2³}
= 1000 0000 0000 1111
↳ ∴ negative ←

∴ we need 2's complement.
⇒ 0111 1111 1111 0001

= -32753

Floating Point Values

How are floating point values represented? (IEEE 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

f) 0 100030000 0110...0

sign: = 1 (∵ positive)

exp: = $2^7 = 128$

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

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

2⁵ 2⁴ 2³ 2² 2¹ 2⁰ . 2⁻¹ 2⁻² 2⁻³ ...
0 1 0 1 1 0 . 0 1 1 0 ...

c) 0 0 111111 1000000 -1

sign: 1

exp: 127

frac: 0.5

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

a) 0 00000000 000...0

sign: 1

exp: 0

frac: 0

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

b) 1 00000000 000...0

sign: -1

exp: 0

frac: 0

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

denormal

in c we're looking mostly

(a) float not double

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

a) $2.5 \div 2^1 = 1.25 \Leftrightarrow 2.5 = 1.25 \times 2^1$
 sign: 1x (bit = 0 to show +ve)
 exp: $1 = \text{exp} - 127 \Rightarrow \text{exp} = 128$
 frac: 0.25_{10}
 bits = 0 1000 0000 0100000000...0

first express the number
in the form:
 $\pm (1 + \text{frac}) \times 2^n$

$$n = \text{exp} - 127$$

b) $0.375 \times 2^2 = 1.5 \Rightarrow 0.375 = 1.5 \times 2^{-2}$
 sign: bit = 0
 exp: $-2 = \text{exp} - 127 \Rightarrow \text{exp} = 125$
 frac: $0.375_{10} = 0.011_2$
 bits = 0 011 1101 01100

frac 128
 64 0
 32 0
 16 0
 8 0
 4 0
 2 0
 1 0
 0 1

1000 0000

c) 27.0
 sign:
 exp:
 frac:
 bits =

next
page.

d) 100
 sign:
 exp:
 frac:
 bits =

to convert frac to
binary
 (x2) 0.25
 0.5
 1.0
 .0

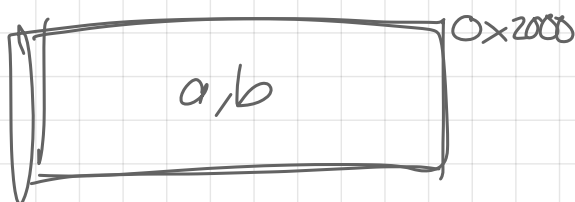
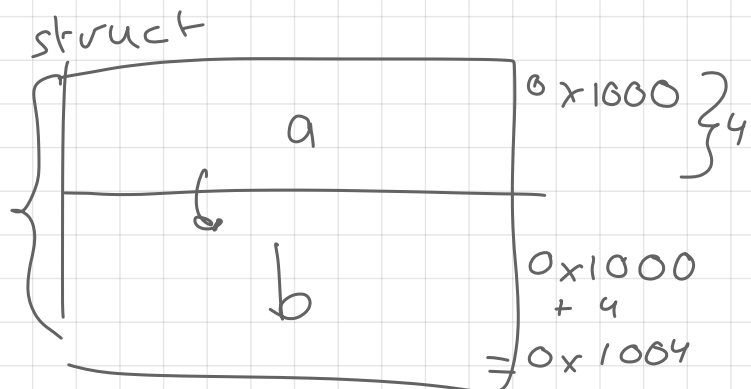
Structs vs Unions

```
struct {
    int a;
    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



$$\begin{array}{r}
 (x2) \quad 0.\overline{375}_{10} \\
 \downarrow \\
 0.75 \\
 1.5 \\
 \hline
 1.0 \\
 \hline
 \end{array}$$

011

$$\begin{aligned}
 27 \div 2^4 &= 1.6875 \\
 \Rightarrow 27 &= (1 + 0.6875) \times 2^{\textcircled{4}} \\
 n &= 4 \quad \swarrow \text{frac}
 \end{aligned}$$

c) 27.0

sign: bit = 0 \because positive

exp: $n = \text{exp} - 127; n=4 \therefore 4 = \text{exp} - 127 \Rightarrow \text{exp } 131_0$

frac: 0.6875 $= 1000\ 0011_2$

bits = 0 1000 0011 10110...0

$$\begin{array}{r}
 (x2) \quad 0.6875 \\
 \downarrow \\
 1.375 \\
 0.75 \\
 1.5 \\
 1.0
 \end{array}$$

Week 5

3. $a = 0x10000020$

b: 24

C:Z 8

 $2c$

d: 30

e: 34

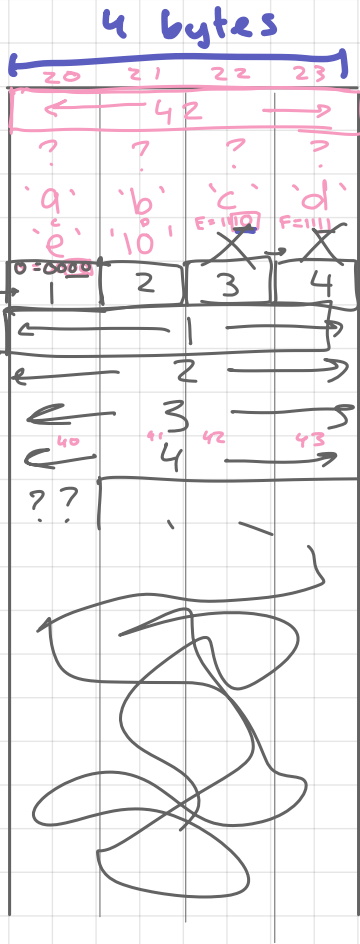
38

3c

40

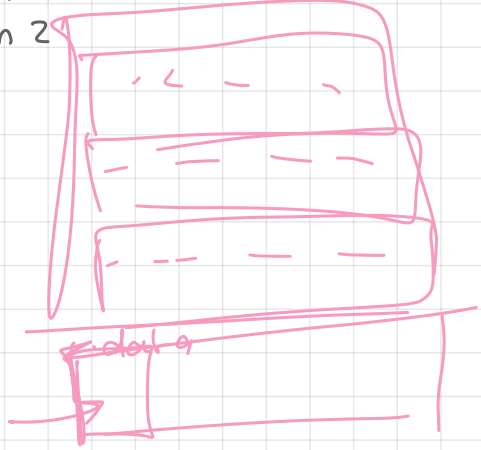
f: 44

1

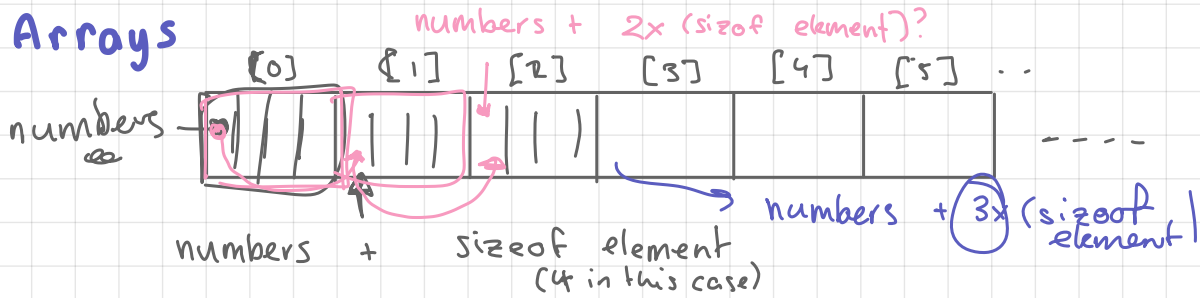


```
int v;
```

eg. `.align 1`
eg. `.align 2`



Arrays



calculate index address:

$$\text{address} = \begin{matrix} \text{start} \\ \text{address} \\ \text{(eg numbers)} \end{matrix} + \text{index} \left(\text{sizeof element} \right)$$

for 2D arrays:

$$\text{address} = \text{start address of row} + \text{col} \times (\text{sizeof element})$$

$$\text{start address row} = \text{sizeof element} \times n\text{-cols} \times \text{row}$$

+ starting address of the array

Week 7

Rights

callee
is function
being called

freedom to use ^{some} registers
however without saving:
\$t, \$a, \$
arguments passed in through
\$a0... \$a3

caller

is the one
doing the
calling

Responsibilities

Save: \$sX, \$ra, \$sp

return values in: \$v0, \$v1
for when value
returning is 4 < n-bytes ≤ 8

The stack



