Pointers and Dynamic memory

Pointers

- holds memory address
  - ↳ points to the location of an object in memory

if they dont have a location, they point to the Null pointer (which is just 0)

remember : memory is stored in registers that be accessed at specific address

- bytes → words
  - ↳ 8 bits or 2 nibbles

pointers point to address they are assigned

↳ using "&"

- mult pointer can point to the same address
- can be deferenced using (*&)
  - ↳ helps manipulate several variables

example:

*& bar = &foo

↳ pointer variable ↳ address of foo is in bar

Pointers can "return" more than 1 value

↳ DAS

- since copies of data aren't ~~be~~ passed in the stack, we can use pointers!

passing by vale = the passed vales are duplicated on the stack

passing by refence = duplicates a pointer onto the stack

->recall cup example!

passing by ref: You simple tell where the data is stored

Arithmetic can be perform on ponters

++: goes to the nex add, increments by 4

--: goes back the add, decrement by 4

+=: add a vale to ancthar ponter

-= distance between both ponter

offset a pointer = +/- an int

-> look up more info about this on the book

-You can't + / or two pointers

ints are typically 4 bytes

arrays can always be written as pointers

- declaring an array in a function allocates it on the stack

data area -> global array

-allocating on the heap

strings are handled as array

String = a pointer to an array of chars

↳ can be indexed or passed by reference

pointers can point to other pointers

↳ char **

Multidimensional Arrays

- can be of any dimension

function pointers

- points to executable code in memory instead of data value
- derefeencing a function pointer yields the referenced functions → parantheses can be around the pointer)

Bit vectors and sets

unit	Size in bits	value	notes
bit	1		
nibble	4	0/1	smallest
byte	8	hex	
half-w	16	ASCII	smallest add
word	32		
L word	64		native size, reg leng
			native size, reg length

Logical Shift

left : shifts the zero to the right

msb -> Lsb ← moves

right : shifts zero to the left

moves -> msb -> Lsb

Arithmetic shifts

left : zeros shifted to the right

msb -> lsb move ←

right : sign bits shifted to the left

msb -> lsb -> moves

Operations

& : and

| : or

~ : not

^ : xor

<< : left shift

>> : right shift

v1 >> v2 : v1 shifted v2 bits

v1 << v2 : v2 is v2 left-shifted v2 bits

And

A	b	c
0	0	0
0	1	0
1	0	0
1	1	1

Or

A	b	c
0	0	0
0	1	1
1	0	1
1	1	1

xor

A	b	c
0	0	0
0	1	1
1	0	1
1	1	0

NOt

~~A~~ b	c
0	1
1	0

higher-order-nibble

- high-order nibble: most significant 4 bits

LSb

msb

$\underbrace{0\ 0\ 1\ 1}_{\text{high-order nibble}}\ 0\ 1\ 1\ 1$

- right shifts 4 times so that the higher-nibble takes the place of low-order nibble
- 3 w10x0f

Setting a high-order nibble

- placing the nibble
- and with 0x0f
- left shift 4 times
- or byte wth bit shifted nibble

sets

- unordeed collections that are characterized by the elements they contain
- iff they have the same elements, they are =

∘ $A \cap B$ = intersection → in both sets

· $A \cup B$ = union → in any sets

∘ $A - B = (A \cap \bar{B})$ → Diff → in A that not in b

∘ C $\bar{A}$ = complement → not in A

sets / Bits

0 = element not a member of the set

1 = element is a member of the set

play around with the set a bit, clear a bit, and get a bit functions!