

Pointers and Dynamic memory

Pointers

- holds memory address

- ↳ points to the location of an object in memory

If they don't 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 dereferenced using (*&)

- ↳ helps manipulate several variables

example:

```
&bar = &foo
```

↳ pointer variable ↳ address of foo is in bar

Pointers can "return" more than 1 value

- ↳ D.A.S

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

Passing by value = the passed values are duplicated on the stack
Passing by reference = duplicates a pointer onto the stack

→ Recall cup example!

Passing by ref: You simply tell where the data is stored

Arithmetic can be performed on pointers
++: goes to the next add, increments by 4
--: goes back the add, decrement by 4
+= add a value to another pointer
-= distance between both pointers

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 arrays

String = a pointer to an array of chars
↳ can be indexed or passed by reference
pointers can point to other pointers
↳ char \Rightarrow

Multidimensional Arrays

- Can be of any dimension

function pointers

- Points to executable code in memory instead of data value

- dereferencing a function pointer yields the referenced functions \rightarrow parentheses can be around the pointer

Bit vectors and sets

Unit	Size in bits	Value	Notes
bit	1	0/1	smallest
nibble	4	hex	
byte	8	ASCII	smallest add
half-word	16		
word	32		
2 word	64		native size, reg length native size, reg length

Logical Shift

left: shifts the zero to the right
msb \rightarrow lsb \leftarrow moves

right: shifts zero to the left
moves \rightarrow msb \rightarrow lsb

Arithmetic Shifts

left: zeros shifted to the right
msb \rightarrow lsb move \leftarrow

right: sign bits shifted to the left
msb \rightarrow lsb \rightarrow 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	0	1
0	1	0
1	0	0
1	1	1

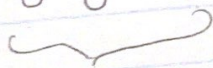
higher-order-nibble

high-order nibble : most significant 4 bits

msb

lsb

0 0 1 1 0 1 1 1



high-order nibble

• right shifts 4 times

so that the

higher-nibble takes

the place of low-

order nibble

0 8 w10xof

Setting a high-order nibble

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

Sets

- Unordered Collections that are characterized by the elements they contain
- iff they have the same elements, they are =
- $A \cap B$ = Intersection \rightarrow in both sets
- $A \cup B$ = Union \rightarrow in any sets
- $A - B = (A \cap \bar{B})$ \rightarrow diff \rightarrow in A that not in B
- \bar{A} = Complement \rightarrow 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!