Section 7
Pointers and Memory

https://xkcd.com/138/

CS 107A, Autumn 2021
Andrew Benson (adbenson@)

Don’t forget to start recording

Unix Tip Spotlight

- History Search
  - Hit <CTRL+R>
  - Start typing, and it'll search your history for matching commands
  - To go to the previous match, hit <CTRL+R> again
  - Press <ENTER> to run, or use arrow keys to edit the command before running
  - Takes some practice before you get used to it, but really really useful

Announcements

- 1:1s
  - Please make my life easier
  - I've pinged anyone on Slack who I don't think has scheduled one yet
- Office Hours
  - Post in #aut21-general if I seem oblivious to your presence
  - In general, @me if you need my attention since I don't get Slack notifications for regular messages

Bird's Eye View

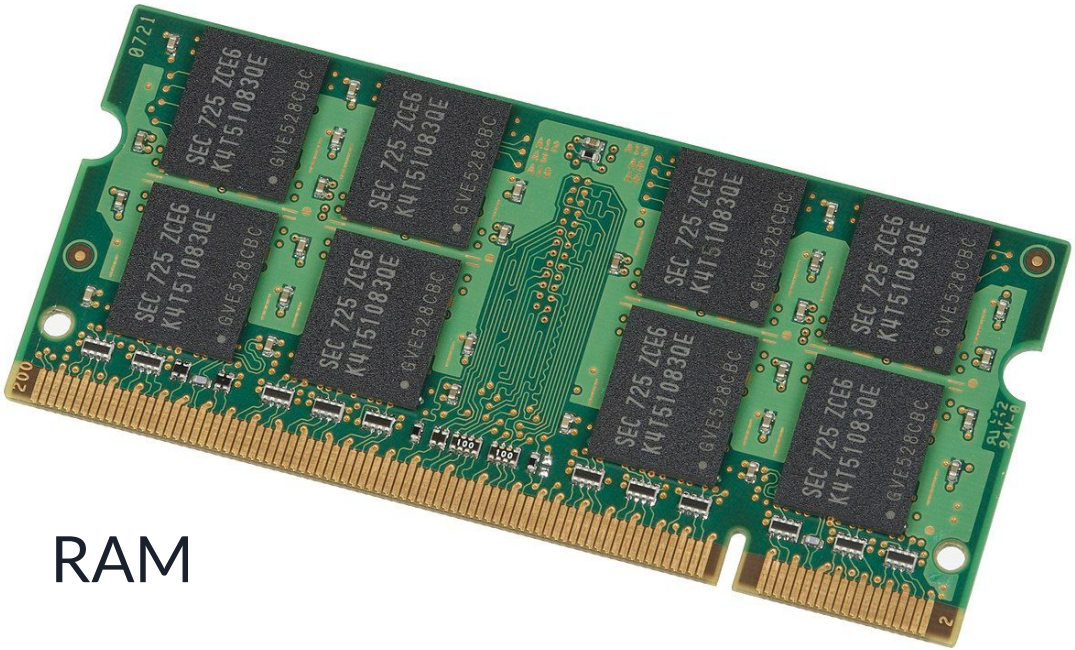
Day	Week 4 Monday	Tuesday	Wednesday	Thursday	Friday	Week 5 Monday	Tuesday	Wednesday
CS 107A		Section: Pointers and Memory		Section: Stack and Heap			Section: Generics	
CS 107	Lecture: Stack and Heap		Lab 3: Arrays / Pointers		Lecture: `void*`, Generics	Lecture: More Generics		Lab 4: `void*` / Function Pointers
CS 107 assignments			assign2 due, assign3 released					assign3 due, assign4 released

Agenda

- From Hardware to the C Memory Model
- Diagramming #1
- Diagramming #2
- Pointer Arithmetic
- Fill-in-the-Blank
- Super Nasty Pointer Problem

From Hardware to the C Memory Model

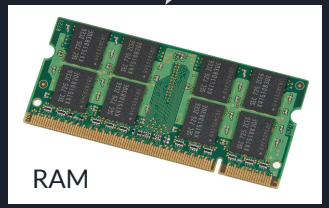
Memory is Hardware



RAM

https://en.wikipedia.org/wiki/Random-access_memory#/media/File:Samsung-1GB-DDR2-Laptop-RAM.jpg

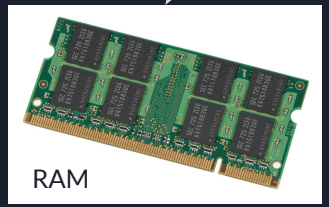
Memory Hardware (RAM) is storage for bits



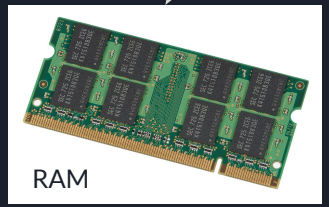

```
11000001010111001010011001100111001000000
00000101101100010111100100110110110110100
01100101110110100111000010010110000011110
01110111001100000100000110010110001001100
00111000111101011010100010100111111111000
11000001010101011111111101111010111011010
10010000101001101101001101110011111011111O
10011000100011011000000010101000101110011
00110110101000001100001001010011010001010
10101001110111010111001110110000001100111
11000100101101101001001010111101111010100
00101110010100101111100011101111010110101
00000100110101010000101011100101101011010
```

CPUs group every 8 bits into 1 byte

11000001
01011100
10100110
01100111
00100000
00000010
11011000
10111100
10011011
01101101
00011001
01110110
10011100
00100101
10000011
11001110
11100110
00001000
00110010



C programs view memory in segments




```
0x1900925: 0b11000001
0x1900924: 0b01011100
0x1900923: 0b10100110
0x1900922: 0b01100111
0x1900921: 0b00100000
0x1900920: 0b00000010
0x190091f: 0b11011000
0x190091e: 0b10111100
0x190091d: 0b10011011
0x190091c: 0b01101101
0x190091b: 0b00011001
0x190091a: 0b01110110
0x1900919: 0b10011100
0x1900918: 0b00100101
0x1900917: 0b10000011
0x1900916: 0b11001110
0x1900915: 0b11100110
0x1900914: 0b00001000
0x1900913: 0b00110010
```

THE STACK

Addresses 0x90beef+ (not shown)

THE HEAP

Addresses 0xdead+ (not shown)

THE DATA SEGMENT

Addresses 0+ (not shown)

RESERVED FOR THE OS

Diagramming #1

getKthChar

```
char getKthChar(char *s, int k) {
    return s[k];
}

int main() {
    char *str = “stanford”;
    char buf[2];
    buf[0] = getKthChar(str, 4);
    buf[1] = getKthChar(str, 8);
    return 0;
}
```

Diagramming #2

foobar

```
void foobar(char **s) {
     (*s)[0] = ‘f’;
     *s = NULL;
     s = NULL;
     // what do things look like right before foobar returns?
}

int main() {
     char str[] = “berkeley”;
     char *ptr = str;
     foobar(&ptr);
     return 0;
}
```

Pointer Arithmetic

Pointer Arithmetic

- In general, pointer arithmetic works in units of the size of what's pointed to (1 for `char*`, 4 for `int*`, etc)
- `char *chrptr = 0x8000;`
- `chrptr--; // now 0x7fff`
- `chrptr += 5; // now 0x8004`
- `int *intptr = 0x8000;`
- `intptr--; // now 0x7ffc`
- `intptr += 5; // now 0x8010`
- You can cast to a different type if you _really_ need to fine-tune this!

Fill-in-the-Blank

writeString

```
// Argument should be a pointer to a string.
void writeString(______param) {
    *param = “filled-in string”;
}

int main() {
    // Declare an array of strings of length 2.
    _________;
    // Pass in the address of the second element.
    writeString(_______);
    return 0;
}
```

Super Nasty Pointer Problem

```
git clone /afs/ir/class/cs107a/WWW/git/section7
```