CMPE-013/L

Introduction to "C" Programming

Max Lichtenstein

Roadmap

- Announcements
- Battleboats Stuff?
  - srand() vs rand()
  - ???
- BREAK
- C in an OS
  - Syscalls
  - File I/O
- Tour of Board.h (?)
- Stringification Macros?
- Software Design Principles (?)

Announcements

- If you have grade issues talk to me now
  - Grades will be finalized on September 4th

srand() vs rand()

rand() model:

- Keep a long list of random-seeming numbers, and an index:

```
List: ['0b101', '0b001', '0b1000', '0b011', '0b110', '0b110', '0b111', '0b010', '0b100', '0b000']
Index:  ^^^^^
```

- Each call to rand() returns current item in list, and advances the index:

```
X = rand();  //x now equals 0b101
```

```
List: ['0b101', '0b001', '0b1000', '0b011', '0b110', '0b110', '0b111', '0b010', '0b100', '0b000']
Index:          ^^^^^
```

```
X = rand();  //x now equals 0b101
```

```
List: ['0b101', '0b001', '0b1000', '0b011', '0b110', '0b110', '0b111', '0b010', '0b100', '0b000']
Index:                   ^^^^^
```

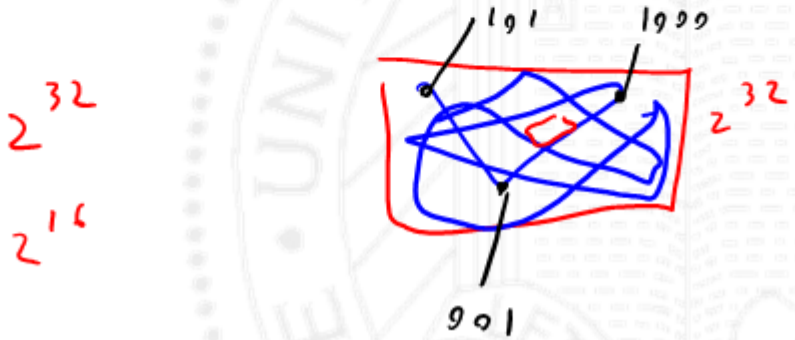
srand() model:

- srand(seed) changes the index to "seed":

```
srand(5);  //now the index points to element 5
```

```
List: ['0b101', '0b001', '0b1000', '0b011', '0b110', '0b110', '0b111', '0b010', '0b100', '0b000']
Index:                                      ^^^^^
```

  - And now future calls to rand() will start at the new index



It's only a model!

- Holding a list that long would be impractical
  - 2^32 entries * 2 bytes/entry ~= 10 GB
  - Also, how do you *make* the list?

It's only a model!

- Instead, use a formula to generate the next number
  - Often this one (linear congruential generator, or LCG):

$$X_{n+1} = (aX_n + c) \mod m$$

  - X is "index" (usually called "state")
  - a, c, m are large, carefully chosen constants
    - In GCC: m = 2^31, a = 1103515245, c = 12345

Max Lichtenstein

Combining randomness

A	B	XOR
0	0	0
1	1	0
1	1	0
1	0	1

A = B

75% of the time

- XORing two random bitstrings
  - If the two sources are independent, this always increases entropy
    - Can combine sources of randomness at no "cost"
    - Except time/power!

A ⊕ 0 = A XOR A ⊕ 1 = ¬A

- On an OS, sometimes there is a shared "entropy pool"
  - On UNIX, /dev/random

Max Lichtenstein

Other Battleboats stuff?

```
Hash(uint16_t A){
  A = A*A;
  A = A^BEEF;
  Return A;
```

A	A*A	A^BEEF
0x0100	0	

32
x 10
320

```
  00000001 00000000
x 00000001 00000000
-------------------
  00000000 00000000
```

```
Hash(uint16 A){
return ((A*A) %. BEEF)
```

Hash(0xBEEF)

Hash(0xBEEF+1)

Type cast

```
return ((uint_64)A *A)) %. BEEF
```

```
Uint64_A BigA = A;
long int
```

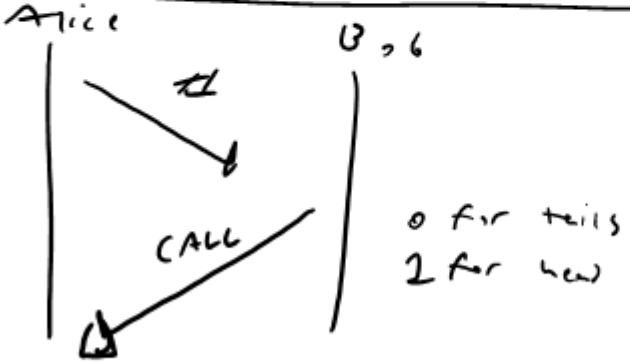
$BEEF^2 * 2 \cdot BEEF * 1^2$

$0 + 2 \cdot 0 + 1 = 1$

"CHH, 1, 3"

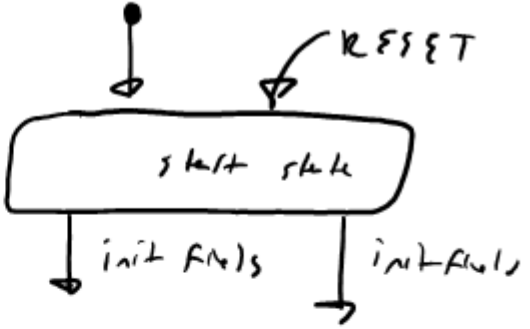
"CHA, 1, 3"

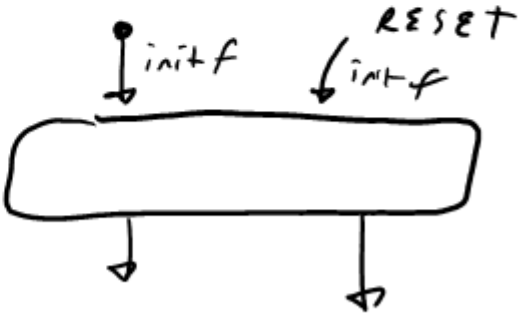
"$CHA, 2, 6E" ← Decode



Agent Init

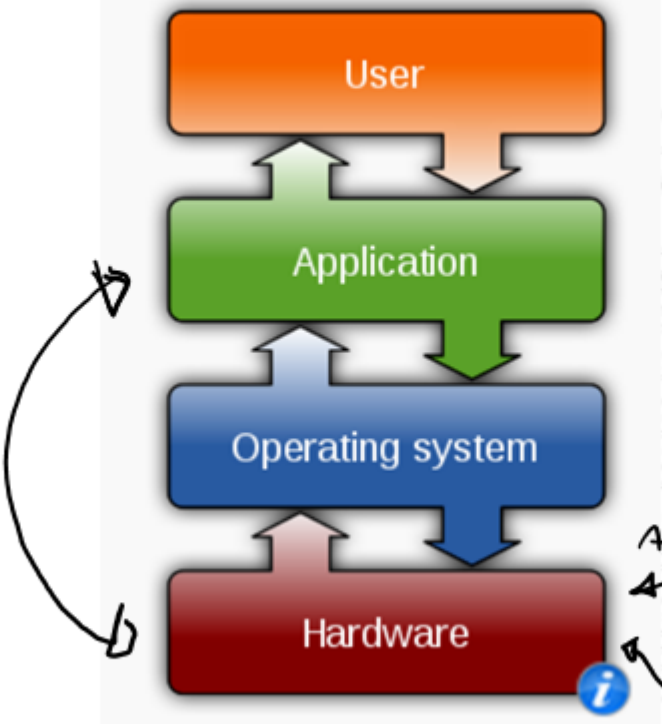

2
2
1
2
Alice
3,6


RESET
start state
init fails
init fails


RESET
init f
init f

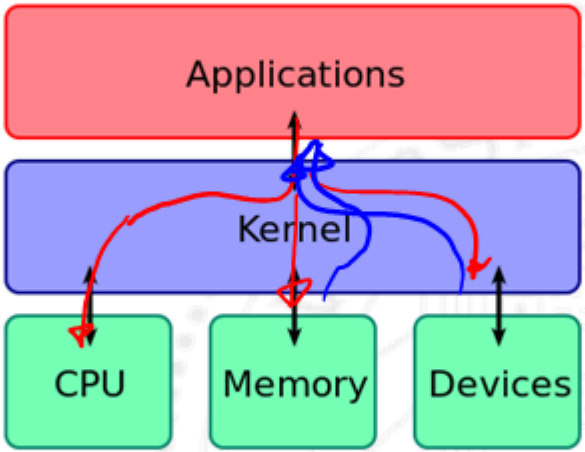
C in an OS

OS model:

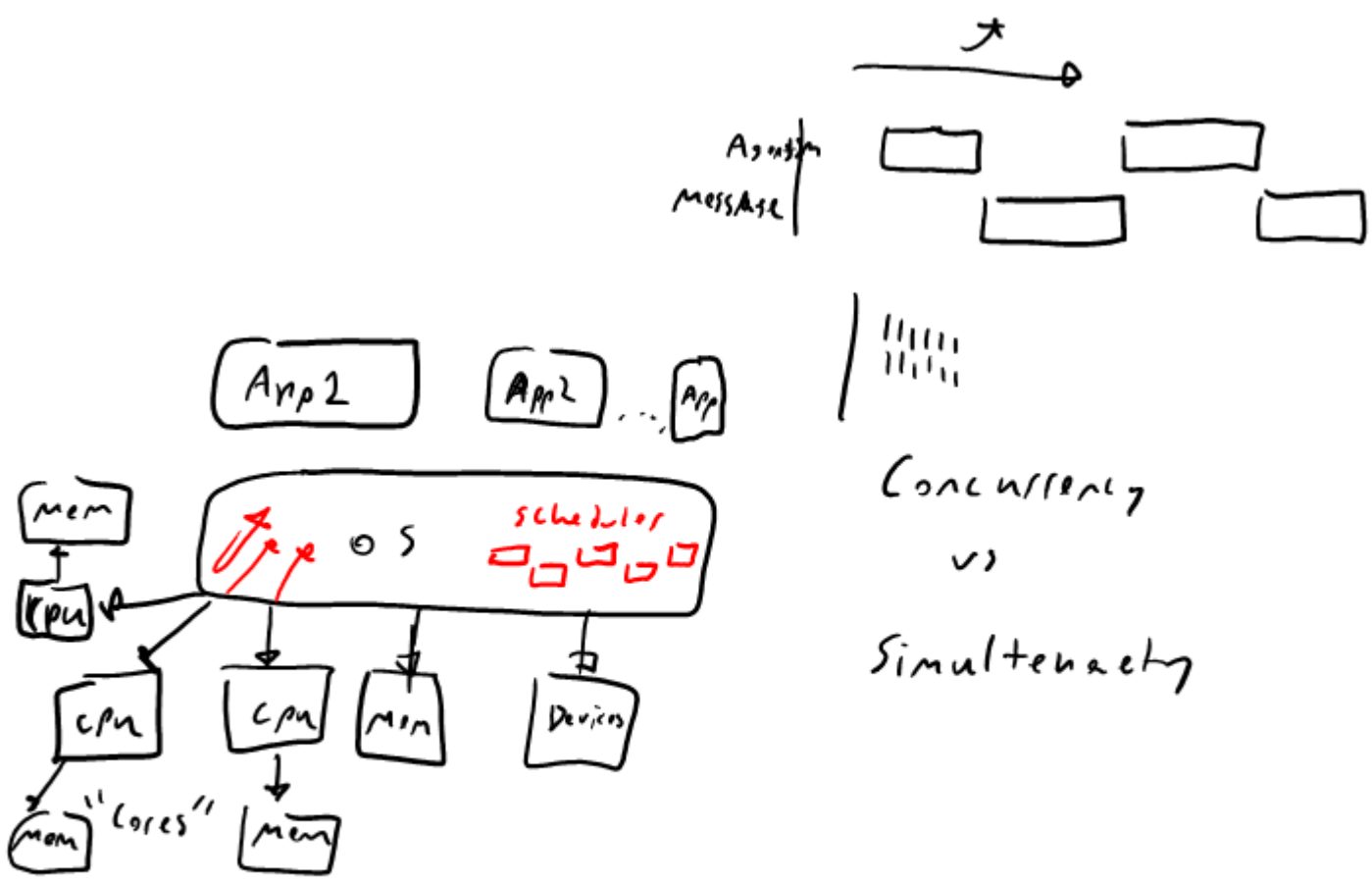


Max Lichtenstein

OS model:



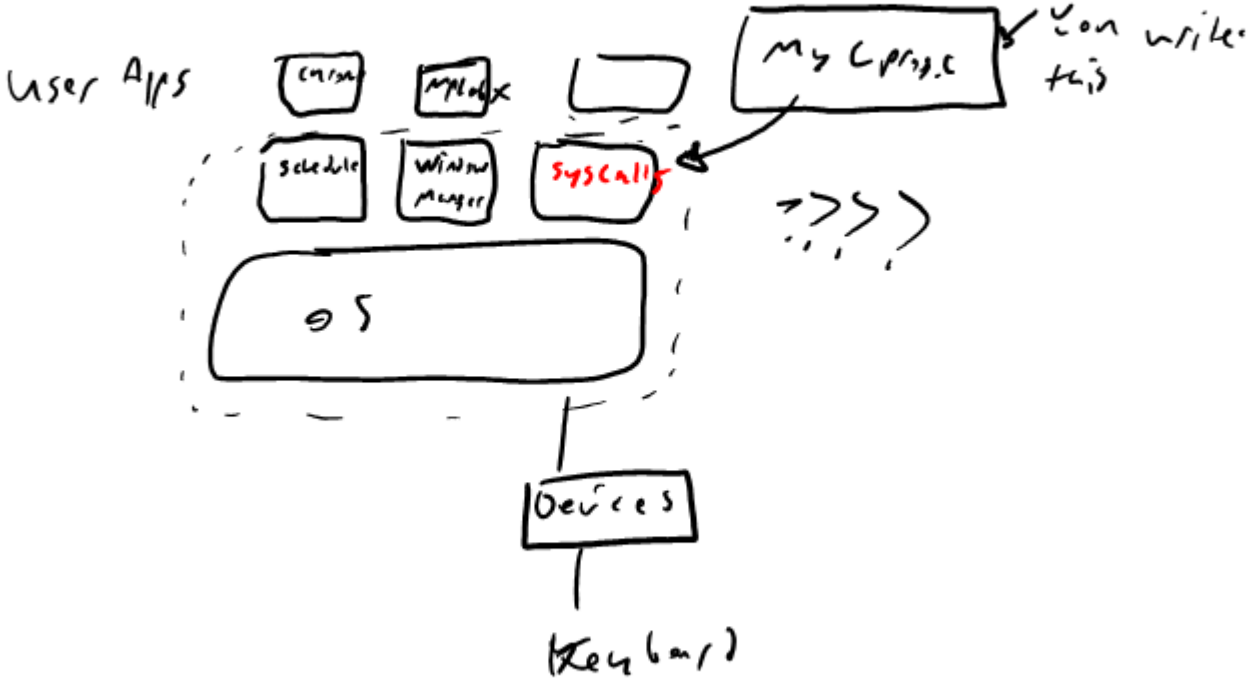
Max Lichtenstein



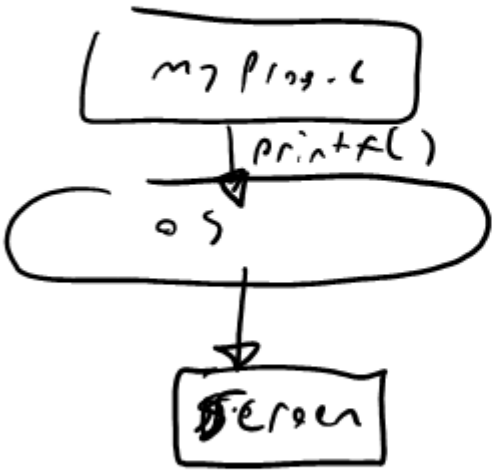
Concurrency

vs

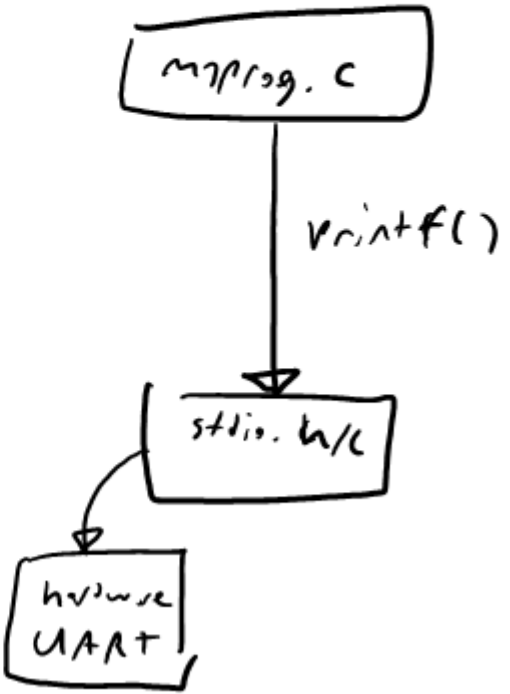
Simultaneity


User Apps
Shell
Editor
Schedule
Window Manager
Syscalls
My Cprog.c
You write this
???
OS
Devices
Keyboard

OS


my prog.c
printf()
OS
Screen

Embedded


myprog.c
printf()
stdio.h/c
hardware
UART

Syscalls:

- Interface between your code and the OS:
  - Process control:
    - Run, check, end other processes
  - File IO
    - Everything is a file! (UNIX)
    - Communicate with user
  - Talk to other devices (USB, etc)
  - Use system resources
    - Time (Real Time Calendar and Clock – RTCC)
    - Open, read, close IP sockets
    - Threading
    - Malloc()

Man Pages:

- Built-in manual
  - So handy! Just call "man XXXX"

```
[mnlichte@unix4 ~]$ man rtc
```

```
RTC(4)          Linux Programmer's Manual          RTC(4)

NAME
       rtc - real-time clock

SYNOPSIS
       #include <linux/rtc.h>

       int ioctl(fd, RTC_request, param);

DESCRIPTION
       This  is  the  interface to drivers for real-
       time clocks (RTCs).

       Most computers  have  one  or  more  hardware
       clocks  which record the current "wall clock"
       time.  These are called  "Real  Time  Clocks"
```

Man Pages:
(divided into sections)

- 1 User Commands
- 2 System Calls
- 3 C Library Functions
- 4 Devices and Special Files
- 5 File Formats and Conventions
- 6 Games et. al.
- 7 Miscellanea
- 8 System Administration tools and Daemons

Man Pages:

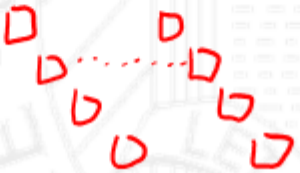
```
[mnlichte@unix4 ~]$ man 3 read
```

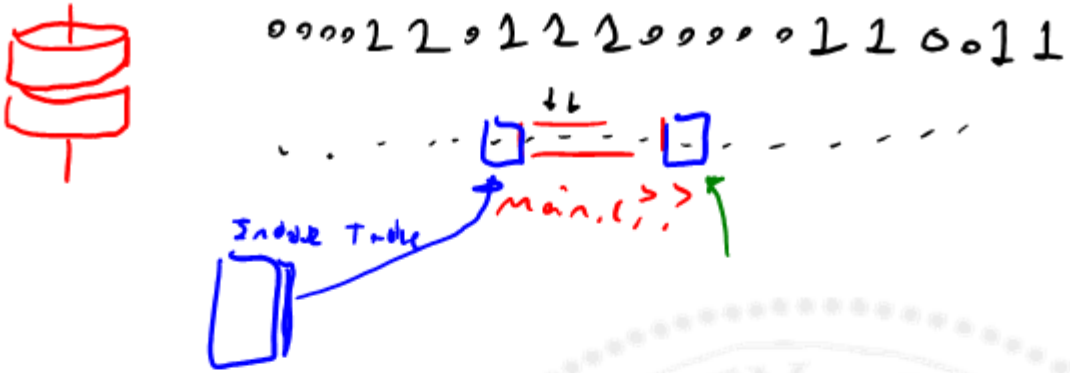
```
READ(3P)      POSIX Programmer's Manual      READ(3P)

PROLOG
       This  manual  page  is part of the POSIX Pro-
       grammer's Manual.  The  Linux  implementation
       of  this  interface  may  differ (consult the
       corresponding Linux manual page  for  details
       of  Linux behavior), or the interface may not
       be implemented on Linux.

NAME
       pread, read - read from a file

SYNOPSIS
       #include <unistd.h>

       ssize_t pread(int fildes, void  *buf,  size_t
       nbyte, off_t offset);
       ssize_t  read(int  fildes,  void *buf, size_t
       nbyte);

DESCRIPTION
       The read() function  shall  attempt  to  read
       nbyte bytes from the file associated with the
       open file descriptor, fildes, into the buffer
```

Max Lichtenstein

More about Syscalls:

- OS-specific
- Implemented in assembly
  - Set a syscall flag
  - but <stdlib.h> and others wrap these subroutines
  - "import time" etc, does the same
- SLOW
  - Calling OS surrenders control of process temporarily
  - Some system resources take milliseconds (SO SLOW)





File I/O

What is a file?

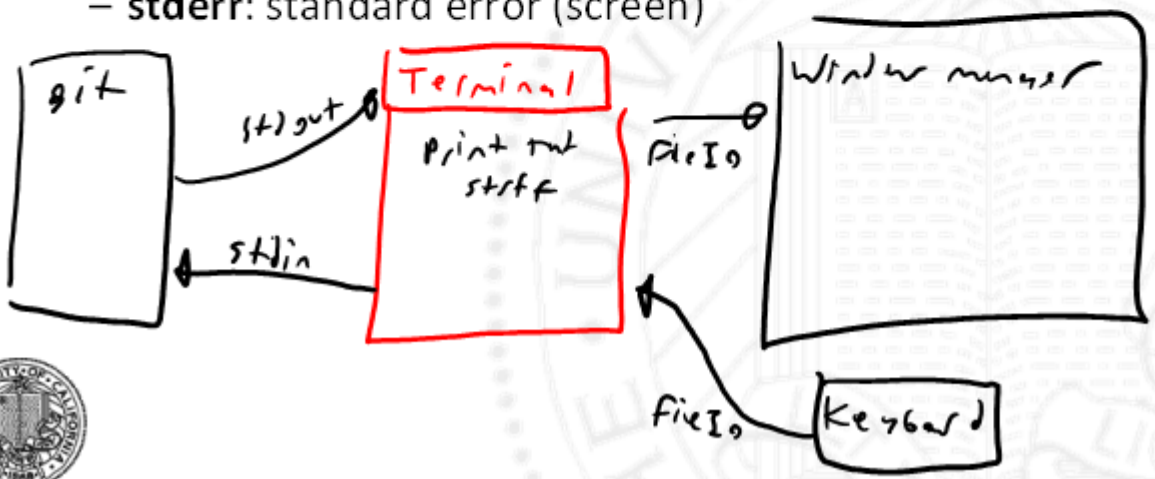
File I/O

- Most data on computers are stored in files
- So accessing data reads and writes to these files
- And in a Unix environment, everything is a file
  - Serial ports
  - Network connections
  - Hard drives
  - Displays
- So everything can be controlled via file access

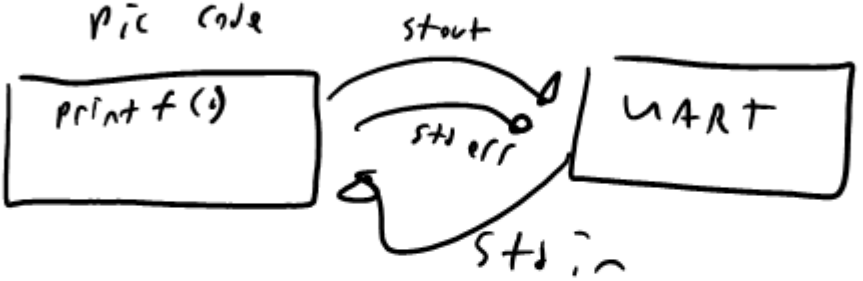
File I/O

Standard files

- Three special files that are automatically opened and closed
  - **stdin**: standard input (keyboard/serial port)
  - **stdout**: standard output (screen)
  - **stderr**: standard error (screen)



<stdio.h>



File I/O

The standard library

- <stdio.h> contains functions for working with files
- Its concept of a file includes:
  - Filename
  - File access mode
  - File size
  - Current position

stdio

File I/O

Using files

- Files are opened with fopen()
- Files are read and written to:
  - fprintf(), fscanf() – Formatting strings
  - fputc(), fgetc() – Characters
  - fputs(), fgets() – Lines
  - **fread()**, fwrite() – Blocks
- Files are closed with fclose()

File I/O

FILE

- The standard library uses a single struct to store the metadata of the file

```
typedef struct _iobuf {
  char *_ptr;
  int _cnt;
  char *_base;
  unsigned short _flag;
  short _file;
  size_t _size;
} FILE;
```

File I/O

fopen()

FILE

Syntax

```
FILE *fopen(const char *name, const char *mode);
```

- ***name*** is a C string with the filename
- ***mode*** is the mode to open the file in
  - "r" opens for reading
  - "w" opens for writing
  - "a" opens for appending
  - "b" specifies binary
- Returns the file pointer

File I/O

File modes

Mode	Meaning
r	Open a text file for reading.
w	Truncate to zero length or create a text file for writing.
a	Append; open or create a text file for writing at the end-of-file .
rb	Open a binary file for reading.
wb	Truncate to zero length or create a binary file for writing.
ab	Append; open or create a binary file for writing at the end-of-file.
r+	Open a text file for read/write.
w+	Truncate to zero length or create a text file for read/write.
a+	Append; open or create a text file for read/write. You can read data anywhere in the file, but you can write data only at the end-of-file.
r+b or rb+	Open a binary file for read/write.
w+b or wb+	Truncate to zero length or create a binary file for read/write.
a+b or ab+	Append; open or create a binary file for read/write. You can read data anywhere in the file, but you can write data only at the end-of-file.

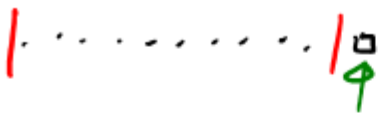
File I/O

fread()

Syntax

```
size_t fread(void *ptr, size_t size,
             size_t count, FILE *stream);
```

- ***ptr*** – The buffer to write into
- ***size*** – The size of each element to read
- ***count*** – The number of elements to read
- ***stream*** – The pointer to the file
- Returns the number of elements read
  - Less than count indicates error or EOF



File I/O

feof()

Syntax

```
int feof(FILE *stream);
```

- ***stream*** – The pointer to the file
- Returns a non-zero value if the stream is at the end of the file, 0 otherwise

```
while (!feof()) {
        getchar(myFILE);
}
```

File I/O

fclose()

Syntax

```
int fclose(FILE *stream);
```

- ***stream*** – The pointer to the file
- Returns 0 if successful, otherwise returns EOF
  - EOF is a macro, generally -1

Example

```
int main(void)
{
  // Open the file, terminating if there was an error
  FILE *pFile = fopen("/room1.txt", "rb");
  if (pFile == NULL) {
    puts("Error opening file.");
    return EXIT_FAILURE;
  }

  // Count the characters in the file.
  int n = 0;
  while (fgetc(pFile) != EOF) {
    ++n;
  }
```

Example

```
// Output the results, if we succeeded
if (feof(pFile)) {
  printf("Total bytes read: %d\n", n);
  fclose(pFile);
  return EXIT_SUCCESS;
}

// Otherwise output an error
puts("Error occurred before reading end of file.");
fclose(pFile);
return EXIT_FAILURE;
```

File formats

File formats

Types

- Two groups:
  - Text
  - Binary
- Text are easier to process, but larger
- Binary are harder to process, but smaller
- Many formats are now compressed (encoded) text files so the data is easy to parse, but the size is small
  - .docx/.xlsx for example

File formats

```
<metaData>
    <messageInfo name = "System Time" pgn = "126992" size = "8">
        <desc>Represents the current data and time</desc>
        <field
        name = "Days since epoch"
        type = "int"
        offset = "16"
        length = "16"
        signed = "no"
        units = "days"
        endian = "little"
        />
    </messageInfo>
    <messageInfo name = "Rudder" pgn = "127245" size = "6">
        <desc>Represents the current rudder position</desc>
        <field
        name = "Position"
        type = "int"
        offset = "32"
        length = "16"
        signed = "yes"
        units = "rad"
        scaling = "0.0001"
        endian = "little"
        />
    </messageInfo>
</metaData>
```

.xml

Max Lichtenstein

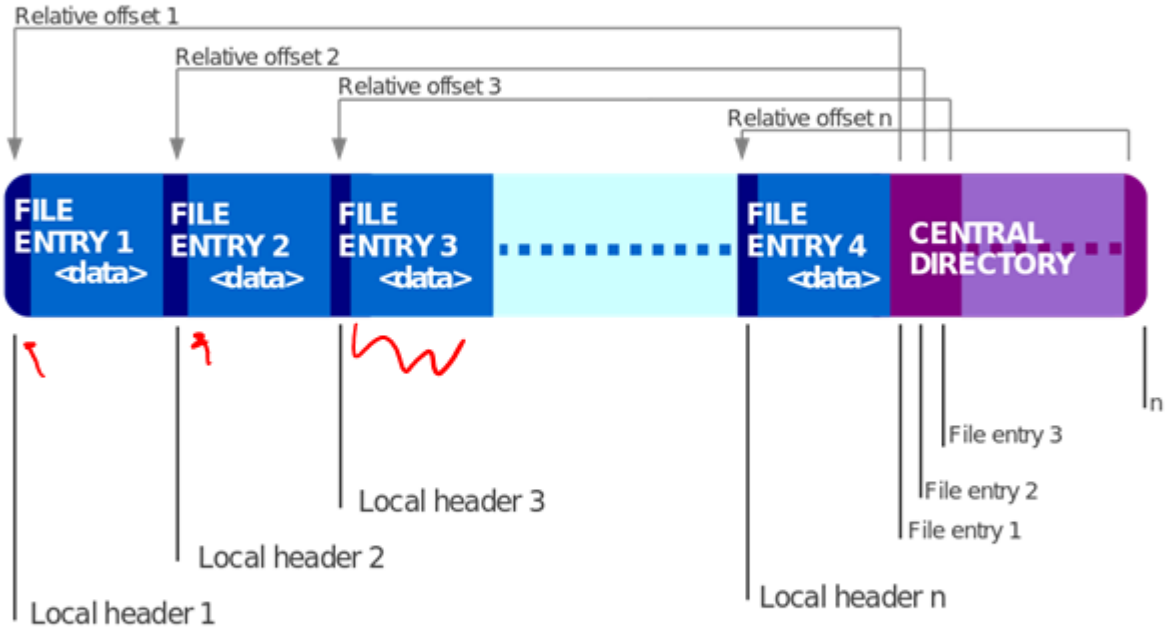
File formats

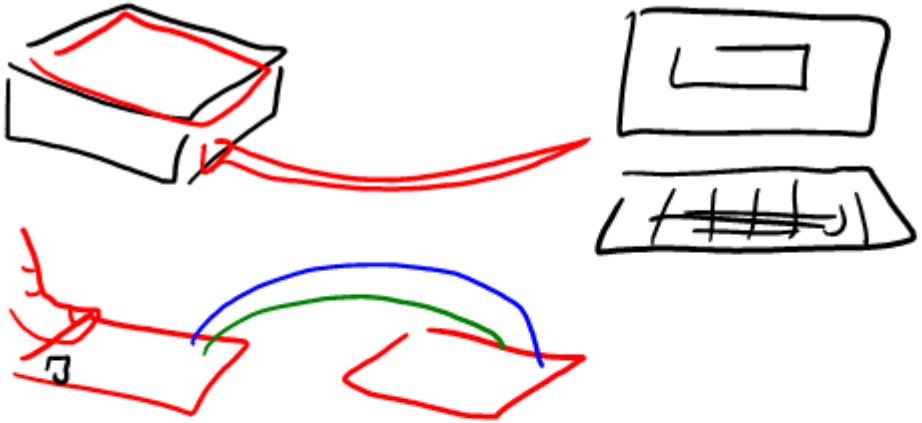
Text: CSV

```
timestamp,time_usec,fix_type,lat,lon,alt,eph,epv,vel,cog
57.300000000000004,57300,450,-59,-15857,0.0,0.0,0.0,57460000,3,369640780,-1220013611,0,150,159,1,13186
57.550000000000004,57550,457,-51,-15855,0.0,0.0,0.0,57760000,3,369640785,-1220013613,0,149,159,1,13411
57.800000000000004,57800,469,-42,-15854,0.0,0.0,0.0,57960000,3,369640786,-1220013615,0,149,159,1,13458
58.050000000000004,58050,474,-32,-15850,0.0,0.0,0.0,58260000,3,369640788,-1220013615,0,149,159,2,13620
58.300000000000004,58300,477,-17,-15847,0.0,0.0,0.0,58460000,3,369640788,-1220013615,0,149,159,2,13620
58.550000000000004,58550,474,-9,-15846,0.0,0.0,0.0,58760000,3,369640793,-1220013616,0,150,159,1,13607
58.800000000000004,58800,469,-12,-15843,0.0,0.0,0.0,58960000,3,369640796,-1220013616,0,149,159,2,13616
59.050000000000004,59050,468,-18,-15839,0.0,0.0,0.0,59260000,3,369640798,-1220013618,0,150,159,2,13486
59.300000000000004,59300,471,-14,-15841,0.0,0.0,0.0,59460000,3,369640798,-1220013618,0,150,159,2,13486
59.550000000000004,59550,485,-4,-15836,0.0,0.0,0.0,59760000,3,369640803,-1220013618,0,149,159,1,13441
59.800000000000004,59800,501,0,-15833,0.0,0.0,0.0,59960000,3,369640804,-1220013618,0,150,159,2,13313
60.050000000000004,60050,502,-18,-15839,0.0,0.0,0.0,60260000,3,369640808,-1220013618,0,150,159,2,13030
60.300000000000004,60300,507,-28,-15839,0.0,0.0,0.0,60460000,3,369640808,-1220013618,0,150,159,2,13030
60.550000000000004,60550,504,-25,-15824,0.0,0.0,0.0,60760000,3,369640815,-1220013620,0,149,159,1,12704
60.800000000000004,60800,515,-20,-15824,0.0,0.0,0.0,60960000,3,369640818,-1220013620,0,150,159,2,12492
61.050000000000004,61050,524,-14,-15832,0.0,0.0,0.0,61260000,3,369640823,-1220013621,0,149,159,1,12492
61.300000000000004,61300,518,-7,-15844,0.0,0.0,0.0,61460000,3,369640823,-1220013621,0,149,159,1,12492
61.550000000000004,61550,512,0,-15825,0.0,0.0,0.0,61760000,3,369640830,-1220013623,0,150,159,5,11498
61.800000000000004,61800,494,0,-15825,0.0,0.0,0.0,61960000,3,369640833,-1220013623,0,150,159,2,11094
62.050000000000004,62050,485,-1,-15824,0.0,0.0,0.0,62260000,3,369640836,-1220013623,0,149,159,1,11094
```

File formats

Binary: ZIP




stuff to

BREAK

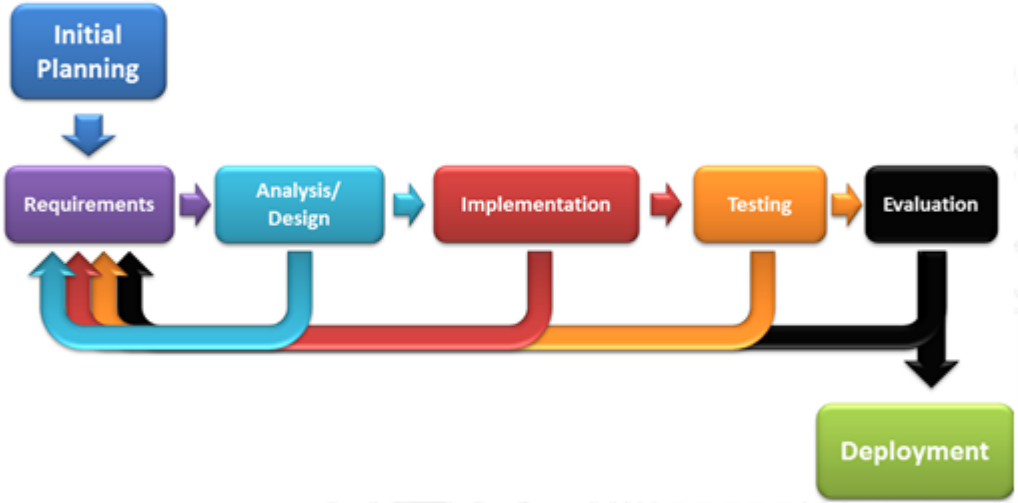
Software Engineering

Design

Build

Software Engineering

Design process



Software Engineering

Principles

- Use consistent styling
- Summary:
  - Utilize whitespace
  - Good variable/function names
  - Comments that describe non-obvious code behavior
    - "How?" and "why?" are good questions to answer in comments

Software Engineering

Principles

- Modularity is important
- Why?
  - Supports code reuse
  - Simplifies changes
  - Allows for testing
- How?
  - Keep functions small
  - Minimize side effects
  - Information hiding/encapsulation

Software Engineering

Principles

- Information hiding/encapsulation
- Summary:
  - Hide unimportant details from the user
  - Protects the user from breaking things
  - Separates backend from frontend

Software Engineering

Mantras

- Keep it simple, stupid
  - KISS
- Summary:
  - Don't solve problems you don't need to
  - Don't introduce unnecessary complexity
  - Prioritize for readability and modularity
  - Don't be clever and/or cute
  - Applies to code architecture and specific code constructs

Software Engineering

Mantras

- Don't repeat yourself
  - DRY
- Summary:
  - Write code only once
  - Simplifies refactoring/incremental development
  - Avoids copy/paste errors

Software Engineering

Mantras

- You aren't gonna need it
  - YAGNI
- Summary:
  - Don't introduce features that are unnecessary
  - Don't write more code then you have to
  - Start small and build from there

Software Engineering

Principles

- Principle of Least Astonishment
- Summary:
  - Be consistent with user's expectations
  - Build on user's intuition
  - Applies to users and developers
    - so both the code and library/program functionality
  - Lowers learning curve

Software Engineering

Principle of Least Astonishment

- Functions/variables should have clear names
  - That should match their functionality!
  - Same for comments
- Functions should not do more than you would think
  - Minimize side effects
- Code should be grouped logically
- Functionality should follow precedence if any exists

Software Engineering

Principles

- Garbage in, garbage out
- Summary:
  - "A system's output quality usually cannot be better than the input quality"
  - So bad input results in garbage output
    - Instead of an error condition
  - Can propagate through the system
  - Can be mitigated by checking the input data

Software Engineering

Principles

- Fault tolerant design
- Summary:
  - Plan for operating failures
    - Running out of memory
    - Data being corrupted
  - Provide fallback modes
  - Important for complex software where minor errors can be common
  - Part of defensive programming

Software Engineering

Principles

- Error tolerant design
- Summary:
  - Plan for user errors
    - "Fault tolerant design" applied to the human component
  - Primarily invalid user input
  - Important for complex software where minor errors can be common
  - Part of defensive programming

Software Engineering

Writing fault/error tolerant code

- Check return values for errors!
  - Many functions have special return values when there are errors, these should usually be checked
  - File accesses
  - scanf()
  - malloc()
- Your code should have special error values
  - LinkedList library
- Program should also return error if failure

Software Engineering

Principles

- Eating your own dogfood
- Summary:
  - When engineers use their own creations, they're generally better
  - More likely that bugs are fixed, features are added because they directly impact the developers
  - In use by all of industry
  - I do it

Software Engineering

Pitfalls

- Premature Optimization
  - "root of all evil"
- Summary:
  - Optimizing code before performance is a critical factor
  - Optimizing reduces readability & modularity
  - Optimization not required for a lot of code
    - See Amdahl's Law
  - See KISS

Software Engineering

Teamwork

- Working as a group is **the** most challenging engineering practice
- Requires:
  - Good communication
- That's it!

Software Engineering

Teamwork

- Pair programming
- Summary:
  - Two developers work side by side: one driving, the other navigating
  - Just like driving:
    - Driver writes code
    - Navigator plans ahead, thinks of edge cases, double-checks driver
  - Requires frequent role switching to be effective!

Software Engineering

Teamwork

- Division of labor
- Summary:
  - Divide work into tasks that can be split between team members
  - Requires coordination to not step on each other's toes
  - Documentation is very important!
  - Can be useful to split testing and development between different people