Arguments & File I/O

Lecture 8
Feb 2nd 2023 | COMP 211-002 | Joshua Bakita

PEARL HACKS FEB 10-12

SCAN ME

- A **free** weekend hackathon for **women and gender non-conforming students**, for participants of all experience levels.
  - *Food + swag*
  - *Career fair*
  - *Social events*
  - *Prizes*
- Centered around mentorship and learning
  - *Workshops*
  - *Networking*
  - *Hands-On Experience*

Welcome!

Today:

- Command Line Arguments
- File I/O

Logistics:

- Midterm study guide coming Friday.
- Midterm review session at 6:20 PM in 014 Sitterson Hall on Tues, Feb. 7th.

Fun fact...

Pearl Hacks

Feb 10-12

Command Line Arguments

Command Line Arguments

`argc` and `argv`

Your main function can optionally receive two arguments, conventionally named `argc` and `argv`.

Command line arguments are delimited by spaces.

`argc` is the number of arguments + 1.

`argv` is a *pointer to an array of char* pointers*.

```
int main(int argc, char* argv[]) {
    /* Your code */
    return 0;
}
```

Functionally identical ways to declare `main()`'s arguments

```
int main(int argc, char** argv) {
    /* Your code */
    return 0;
}
```

Command Line Arguments

Printing our arguments

- ➔ Let's step through this line by line

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char** argv) {
    for (int i = 0; i < argc; i++) {
        int arg_len = strlen(argv[i]);
        for (int j = 0; j < arg_len; j++) {
            putchar(argv[i][j]);
        }
        putchar('\n');
    }
}
```

Try it yourself!

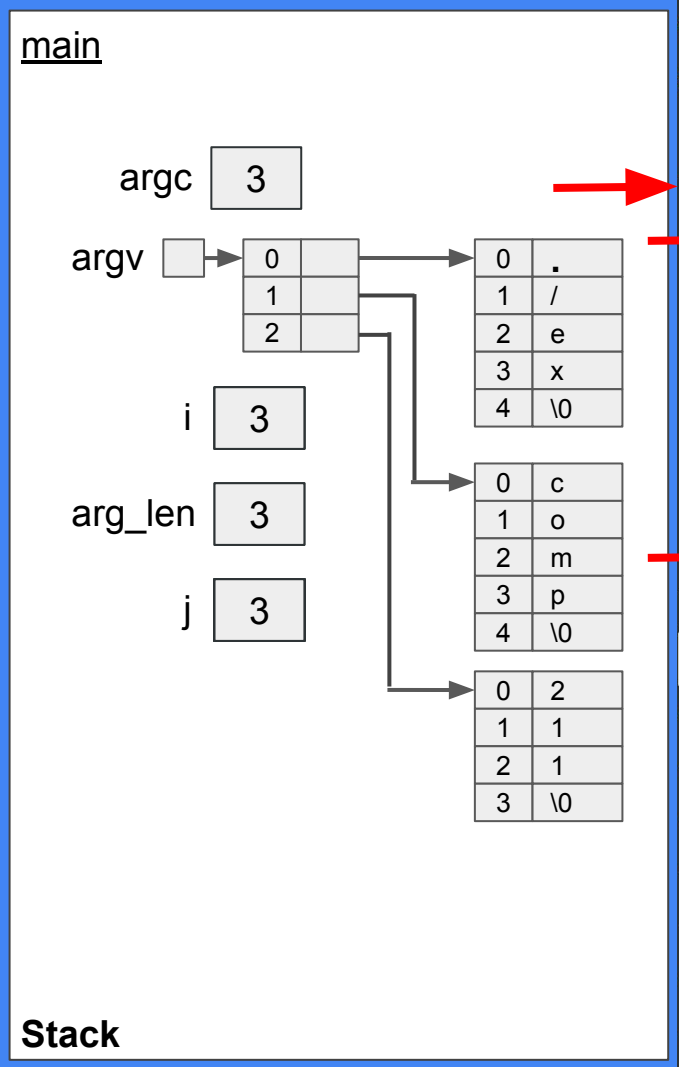
```
$ wget https://www.cs.unc.edu/~jbakita/teach/comp211-s23/l8/example.c
$ gcc example.c -o ex
$ ./ex comp 211
```

Stack

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char** argv) {
    for (int i = 0; i < argc; i++) {
        int arg_len = strlen(argv[i]);
        for (int j = 0; j < arg_len; j++) {
            putchar(argv[i][j]);
        }
        putchar('\n');
    }
}
```

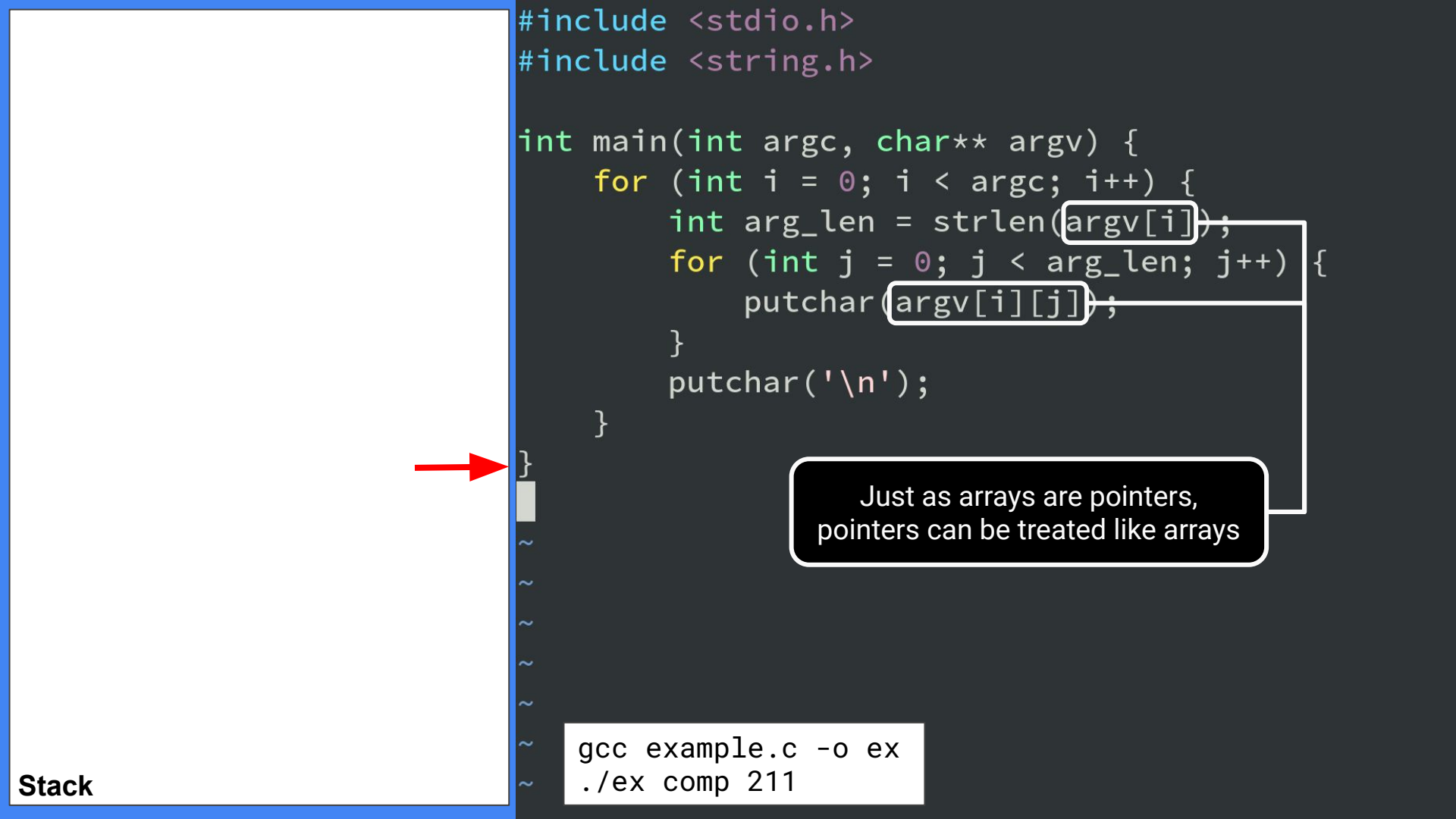
```
gcc example.c -o ex
./ex comp 211
```




```c
#include <stdio.h>
#include <string.h>

int main(int argc, char** argv) {
    for (int i = 0; i < argc; i++) {
        int arg_len = strlen(argv[i]);
        for (int j = 0; j < arg_len; j++) {
            putchar(argv[i][j]);
        }
        putchar('\n');
    }
}
```

```
gcc example.c -o ex
./ex comp 211
```


#include <stdio.h>
#include <string.h>
int main(int argc, char** argv) {
for (int i = 0; i < argc; i++) {
int arg_len = strlen(argv[i]);
for (int j = 0; j < arg_len; j++) {
putchar(argv[i][j]);
}
putchar('\n');
}
}
Just as arrays are pointers, pointers can be treated like arrays
gcc example.c -o ex
./ex comp 211
Stack

```c
// Get the current time in miliseconds
uint64_t get_ms();

int main(int argc, char** argv) {
    if (argc >= 2 && (strcmp(argv[1], "--help") == 0 || strcmp(argv[1], "-h") == 0)) {
        fprintf(stderr,
                "Usage: %s [s(equential)/r(andom)] [number of iterations]\n",
                argv[0]);
        fprintf(stdout,
                "Program will iterate forever if the number of iterations is not specified.\n")
        return 1;
    }

    int is_seq = argc >= 2 ? argv[1][0] != 'r' : 1;

    unsigned long iterations = 0;
    if (argc >= 3) {
        char* status = {'\0'};
        iterations = strtol(argv[2], &status, 10);
        if (iterations == LONG_MIN || iterations == LONG_MAX || status[0] != '\0') {
            perror("Invalid iteration count");
            return 1;
        }
    } else {
        fprintf(stdout, "Infinitely generating %s memory bus traffic...\n",
```

This is an expression with a ternary operator

Aside: Ternary Operator

2.11 Conditional Expressions

The statements

```
if (a > b)
    z = a;
else
    z = b;
```

compute in `z` the maximum of `a` and `b`. The *conditional expression*, written with the ternary operator "`?:`", provides an alternate way to write this and similar constructions. In the expression

expr$_1$ `?` *expr*$_2$ `:` *expr*$_3$

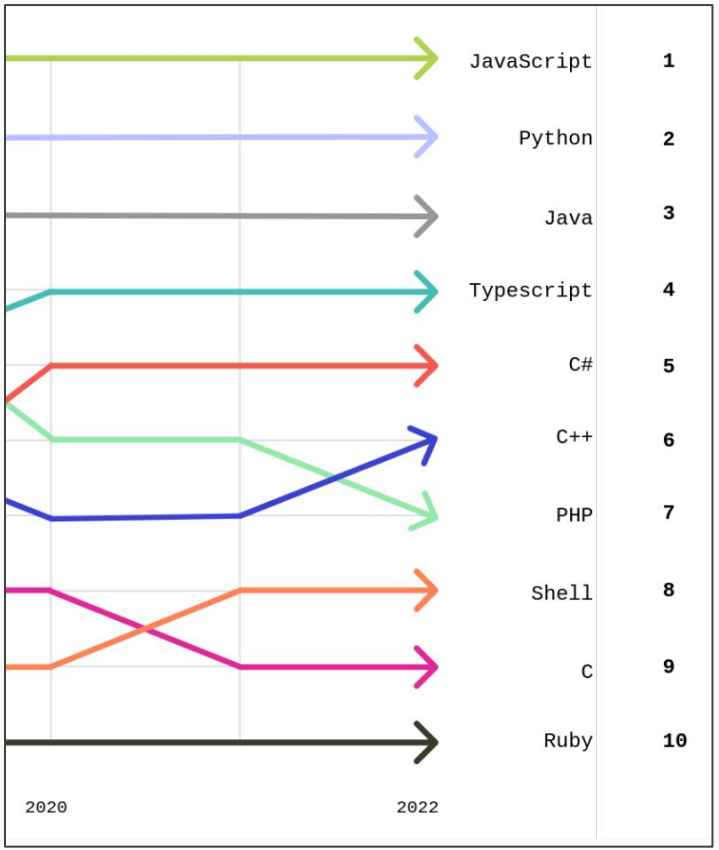
the expression *expr*$_1$ is evaluated first. If it is non-zero (true), then the expression *expr*$_2$ is evaluated, and that is the value of the conditional expression. Otherwise *expr*$_3$ is evaluated, and that is the value. Only one of *expr*$_2$ and *expr*$_3$ is evaluated. Thus to set `z` to the maximum of `a` and `b`,

```
z = (a > b) ? a : b;     /* z = max(a, b) */
```

It should be noted that the conditional expression is indeed an expression, and it can be used wherever any other expression can be.

Sec. 2.11 of *The C Programming Language* ("K&R C")

Top Programming Languages (GitHub)



Return the maximum of a and b

```
a > b ? a : b;
```

```
a if a > b else b
```

```
a > b ? a : b;
```

```
a > b ? a : b;
```

```
a > b ? a : b;
```

```
a > b ? a : b;
```

```
a > b ? a : b
```

```
a > b ? a : b
```

```
a > b ? a : b
```

```c
// Get the current time in miliseconds
uint64_t get_ms();

int main(int argc, char** argv) {
    if (argc >= 2 && (strcmp(argv[1], "--help") == 0 || strcmp(argv[1], "-h") == 0)) {
        fprintf(stderr,
                "Usage: %s [s(equential)/r(andom)] [number of iterations]\n",
                argv[0]);
        fprintf(stdout,
                "Program will iterate forever if the number of iterations is not specified.\n")
        return 1;
    }

    int is_seq = argc >= 2 ? argv[1][0] != 'r' : 1;

    unsigned long iterations = 0;
    if (argc >= 3) {
        char* status = {'\0'};
        iterations = strtol(argv[2], &status, 10);
        if (iterations == LONG_MIN || iterations == LONG_MAX
            perror("Invalid iteration count");
            return 1;
        }
    } else {
        fprintf(stdout, "Infinitely generating %s memory bus traffic...\n",
```

What will the value of is_seq be if I run my program as ./thrasher -r?

https://PollEv.com/joshuabakita182

Grab these slides from the website to see the text up close.

File I/O

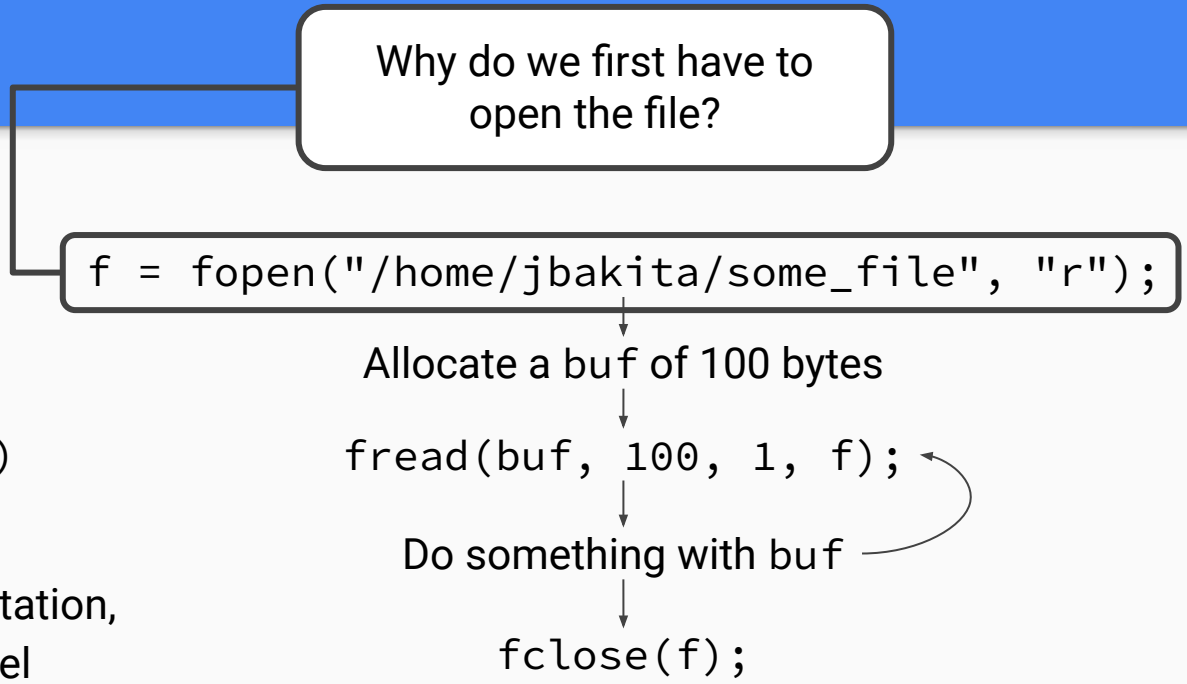
An important part of Assignment 2

Overview

Key functions:

- ➔ Open a file: `fopen()`
- ➔ Read bytes: `fread()`
- ➔ Write bytes: `fwrite()`
- ➔ Move index in file: `fseek()`
- ➔ Close a file: `fclose()`

See the `man` pages for documentation, and the readings for a higher-level description.



Why open a file first?

- ➔ Finding *where* the bytes for a file are can be very involved.
- ➔ Best to only do it once.

1. Traverse the Directory Tree

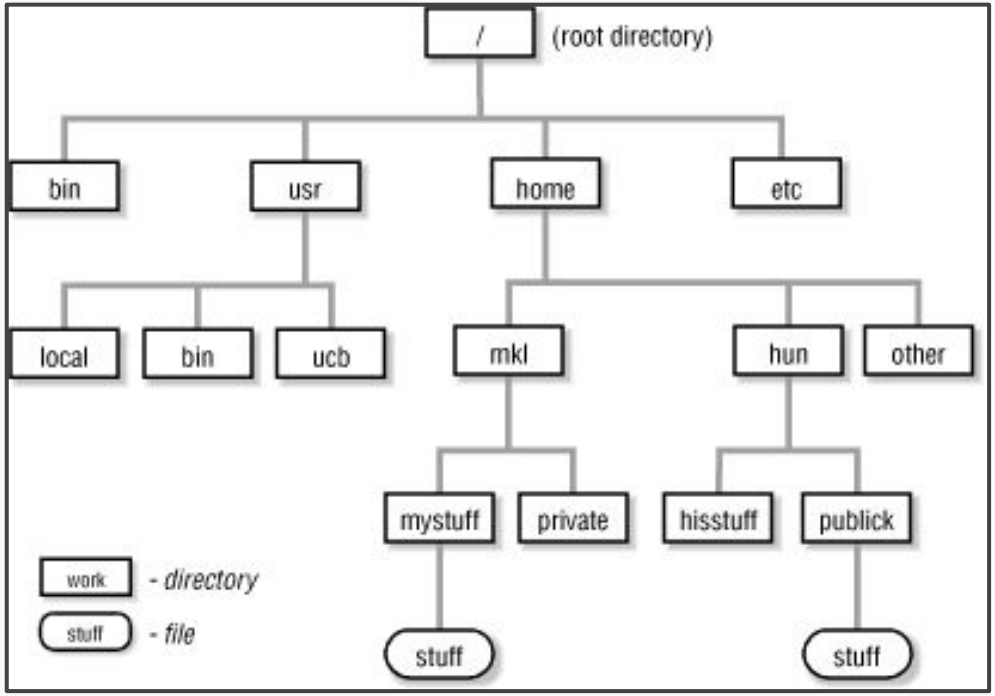
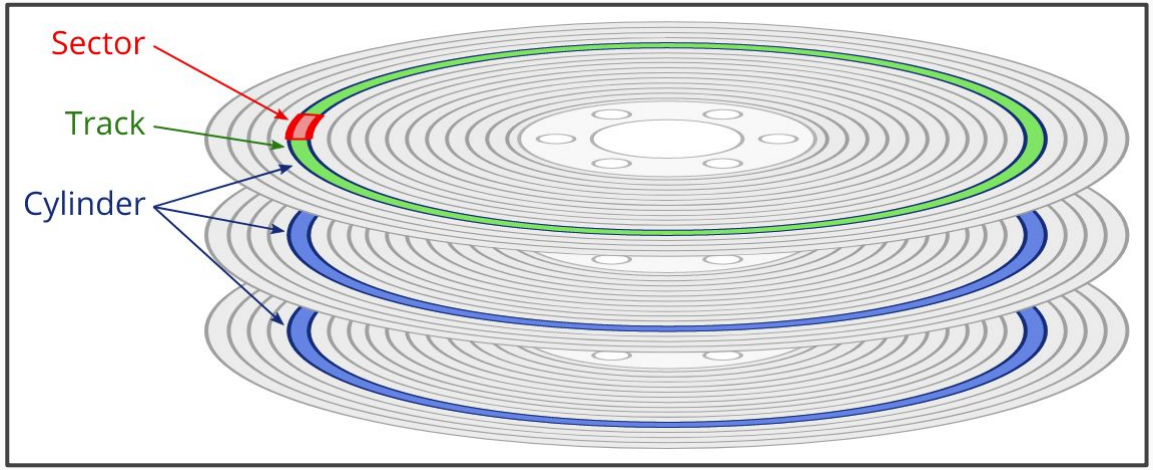


Fig. 1-3 of *UNIX Power Tools, 3rd Ed.*

Why open a file first?

- ➔ Finding *where* the bytes for a file are can be very involved.
- ➔ Best to only do it once.

2. Translate into a storage location

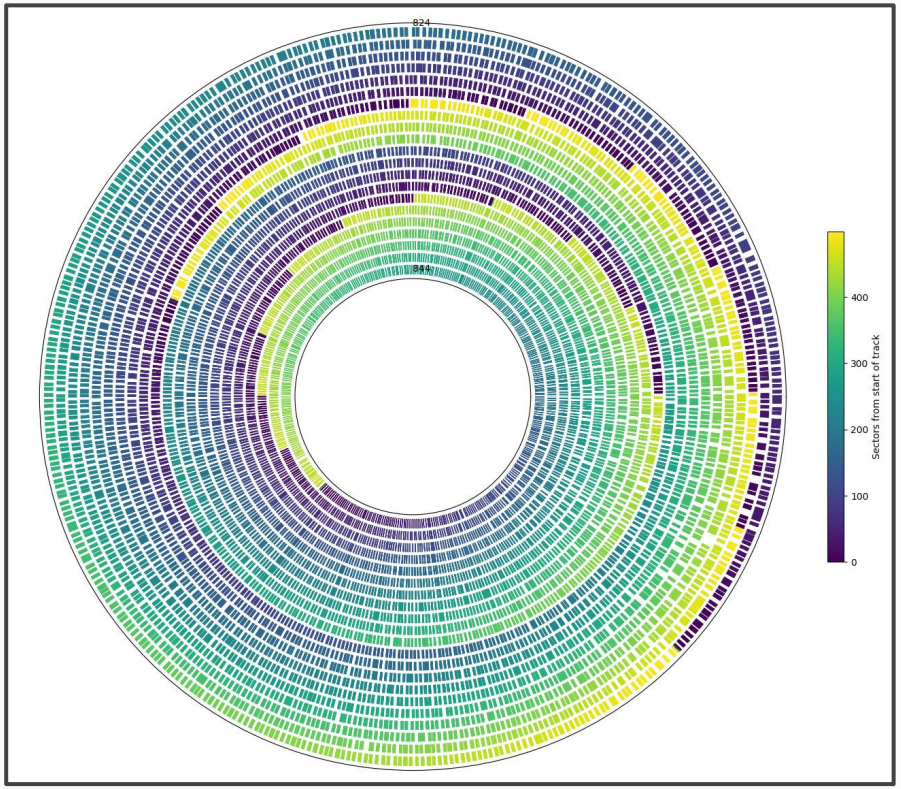


Henry Wong, *Discovering Hard Disk Physical Geometry through Microbenchmarking*

Why open a file first?

- ➔ Finding *where* the bytes for a file are can be very involved.
- ➔ Best to only do it once.

3. Seek to the sector



Henry Wong, *Discovering Hard Disk Physical Geometry through Microbenchmarking*

You've already been using files!

Three open `FILE*` are automatically provided:

- ➔ `stdout`—Standard out
- ➔ `stderr`—Standard error
- ➔ `stdin`—Standard input

`fprintf(stdout, "Hello world\n")` is identical to `printf("Hello world\n")`.

File I/O

Let's build `cat`

Questions?

See office hour calendar on the website for availability.

Contact:
Email: hacker@unc.edu
Twitter: @JJBakita
Web: https://cs.unc.edu/~jbakita

Old Well, University of North Carolina at Chapel Hill, Winter 2017