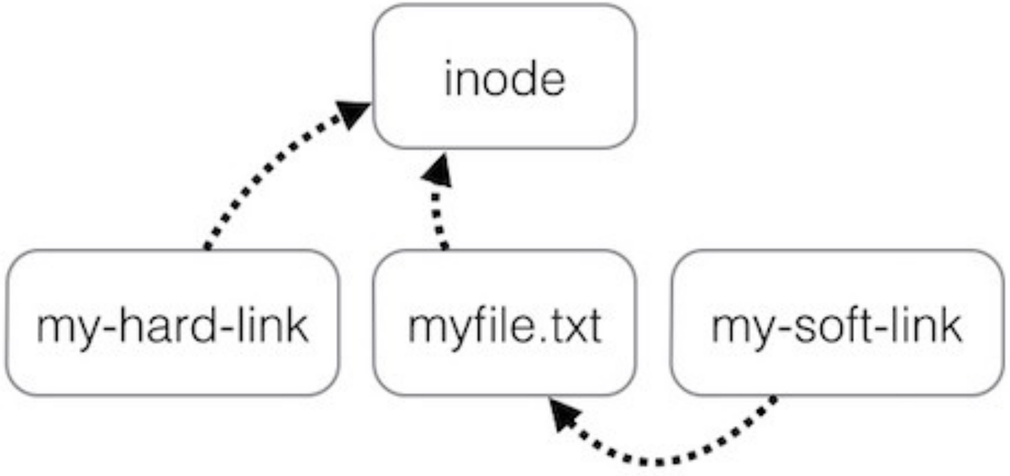
# CS246
# links and the file system
# Arrays and Strings

Feb 25

# Lab — 2 tables of chars

```c
#include <stdio.h>
#define ASCII "%6d%6c\n" // dec, char
#define ALPHABET "%6d%6c%6c\n" // num, uppercase, lowercase
#define UPPERSTART 65 // ascii for 'A'
#define LOWERSTART 97 // ascii for 'a'

int main() {

    for (int i = 33; i < 127; i++) {
        printf(ASCII, i, (char)i);  //the cast to char is unnecessary
    }

    for (int j = 0; j < 26; j++) {
        printf(ALPHABET, j, (char)j+UPPERSTART, (char)j+LOWERSTART);
    }

}
```

# Files and Hard/Soft Links



- in addition to files and
- directories, Unix has “links”
- Hard links
    - only to files
    - only within file systems
    - effectively creates a second user of disk space
- Soft links
    - files and directories
    - links to the file itself, not the disk space
        - dangling links

# More on links

- ln [-s] filename linkname
- ls -l
- ls -i – show the inode
  - disk block locations of the file
- Change source file  both hard and soft have the change
- Overwrite source with new file of same name
  - hard – old file!
  - soft – new file
- Delete aaaa
  - hard – No change!
  - soft – dead link!
- Generally you want soft links

Create a file named “aaaa” and put from keyboard into it

```
gtowell@mil:~$ cat > aaaa
this is a test
gtowell@mil:~$ ln aaaa aaaa_hard
gtowell@mil:~$ ln -s aaaa aaaa_soft
gtowell@mil:~$ ls -l aaaa*
-rw-r--r-- 2 gtowell faculty 15 Feb 19 18:49 aaaa
-rw-r--r-- 2 gtowell faculty 15 Feb 19 18:49
aaaa_hard
lrwxrwxrwx 1 gtowell faculty  9 Feb 19 18:49 aaaa
-> aaaa_soft
gtowell@mil:~$ ls -i aaaa*
169088825 aaaa  169088825 aaaa_hard
169088827 aaaa_soft
gtowell@benz:~$ echo "a new test" > aaaa
gtowell@benz:~$ cat aaaa_hard
this is a test
gtowell@benz:~$ cat aaaa_soft
a new test
gtowell@mil:~$ rm aaaa
gtowell@mil:~$ cat aaaa_hard
thatt this is a test
gtowell@mil:~$ cat aaaa_soft
cat: aaaa_soft: No such file or directory
```

Create a file named “aaaa” and put echoed stuff init

# Typical Unix directories

- / the beginning - the root
- /bin — executables
- /home — user directories
- /lib — libraries
    - parts of executables
    - usually a .so extension  eg libc.so
        - this is the library that from “gcc -lc -xc xxx.c”
- /usr —
    - things that are also in /
        - /usr/bin, /usr/include, …
    - /usr/local — stuff NOT in standard UNIX …
- /proc
    - NOT actual files
        - /proc/cpuinfo, /proc/stat

# stdout and stderr

- Like Java, programmers can choose to write to either stdout or stderr.
  - stdout
    - buffered, lower “cost”
    - Output may get lost
    - pipe-able
  - stderr
    - guaranteed delivery
    - CPU interrupt
    - 2>, NOT pipe-able

BuffTest.c

```c
int main(int argc, char *argv[]) {
    if (argc < 3) {
        printf("Usage: xxx 0_or_1  integer\n");
        return 0;
    }
    FILE *ff = atoi(argv[1]) == 0 ? stdout : stderr;
    int loop = atoi(argv[2]);
    for (int j = 0; j < loop; j++)
    {
        for (int i = 0; i < 26; i++) {
            putc('a' + i, ff);
        }
    }
    int a = 0;
    int c = 5 / a;
    return 0;
}
```

Ternary operator!!!

write one character to an output stream

# Arrays

- char arr[100];
- Can be created from user input
  - int j = atoi(argc[1]);
  - char arr[j];
- may
  - be passed in to functions
  - change values in the array inside functions
- may NOT
  - be created in function then returned
  - Overwrite

See ArrTest.c

# Arrays

sizeof func gives number of bytes of space used

```
double  globalDArr[500];
int main(int argc, char const *argv[])
{
    int j = atoi(argv[1]);
    int arr[j];
    printf("local %d  %d  %d\n", sizeof(arr), sizeof(arr[0]), sizeof(arr)/sizeof(arr[0]));
    printf("Global %d %d  %d\n", sizeof(globalDArr), sizeof(globalDArr[0]), sizeof(globalDArr)/
sizeof(globalDArr[0]));
```

# More Arrays

```
void funcc(int farr[]) {
    printf("BBB %d  %d\n", sizeof(farr), sizeof(farr[0]));
}

/**
 * The preferred way to passing an array.  In this, the link between
 * the integer passed as the size of the array, and the size of the
 * array is explicit
 * arrSize -- the number of items in the array
 * * farr the array
 * **/
void funcc2(int arrSize, int farr[arrSize]) {
    printf("CCC %d  %d\n", sizeof(farr), sizeof(farr[0]));
}
```

# More Arrays — things that do NOT work

```
char[] f3(int n) {
    char arrInFunc[n];
    printf("DDD %d  %d\n", sizeof(arrInFunc), sizeof(arrInFunc));
    return arrInFunc;
}
char* f3a(int n) {
    char arrInFunc[n];
    arrInFunc[0] = 'a';
    printf("DDD %d  %d  %c\n", sizeof(arrInFunc), sizeof(arrInFunc), arrInFunc[0]);
    return (char *)arrInFunc;
}
char globalArr[10]; // a global array to be replaced (NOT)
void f4(int siz) {
    char zzz[siz];
    printf("EEE %d  %d\n", sizeof(zzz), sizeof(zzz));
    globalArr = zzz;
}
```

# So what is an array in C

- int arr[10]
    - causes allocation of space for 10 integers
    - space is contiguous
    - what actually is stored in arr is just “pointer” to beginning of that block of memory
        - where it is knowable, C retains size of memory block
            - knowable within function for arrays allocated in function
            - knowable everywhere for globals
                - because cannot replace
    - So when passed into function all function gets is a pointer

# Java and C

- Java
    - Java arrays are a pointer to a block of memory + size of the memory block
    - “new” operation in java allocates from “heap”
        - heap is global memory space
        - So array allocated in a function can be used outside that function
        - Size of array is bounded by machine memory
- C
    - arrays are pointer to a block of memory
    - global arrays are allocated from heap
    - arrays in functions are allocated in “stack” space
        - space space clears when function completes
        - size of array inside function is bounded by size of “stack space”

# Random Numbers

- Two parts
  - seed `void seed(int)`
  - fetch `int rand(void)`
    - `return value is 0..RANDMAX`
- Think of random number generator as being a really long array of random numbers
  - seed determines location in list
  - rand gets item at current location and increments location.
- So, given the same seed you get the same sequence of “random” numbers
- Problem: to get random numbers you need an unpredictable seed.
  - Common solution use time (or a function thereof)

```
//ArrTest2.c

#include <stdlib.h>
#include <time.h>

time_t tt;
srand(time(&tt));

int r = rand()



Q: How to get rand in 0..N?
Q: How to get rand in range?
```

# Strings

- Formally C does not have strings
- Practically
  - strings implemeted as array of char
  - To be treated as a string the last non-nil character in the array must be ‘\0’
  - so a C string is an array of char that ends in ‘\0’

  - functions for dealing with strings
    - #include <string.h>

# SHOUTj.c

- Capitalize letters in a string
- Doing it like java with lots of functions

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define LINE_LEN 256
#ifndef NULL
#define NULL (void *) 0
#endif
void shout() {
    char line[LINE_LEN];
    while (1)  {
        if (NULL == fgets(line, LINE_LEN, stdin))
            break;
        for (int ptr = 0; ptr < strlen(line); ptr++) {
           if (isalpha(line[ptr])) {
                line[ptr]=toupper(line[ptr]);
            }}
        printf("%s\n", line);
    }}
int main(void) {
    shout();
}
```

from stdio.h

from string.h

from ctype.h

# Lab

- Modify SHOUTj.c
  - do not include either string.h or ctype.h
  - Use ASCII values of chars to determine what to upcase
  - stop going across the array at ‘\0’