ex. Program to copy a file [user will specify file in cmd line]

```
$./copy file1 file2
```

PROGRAM needs to check the cmd args or else it will crash!

```
#include <stdio.h>
int main (int argc, char * argv []) {
    FILE *ifp, *ofp;

    if (argc != 3) {
        fprintf(stderr, "usage: %s [source] [destination] \n",
        return 1;                                    argv [0]);
    }
    if ((ifp = fopen (argv [1], "rb")) == 0 ) {
        perror ("fopen");
        return 2;
    }
    if ((ifp = fopen(argv [2], "wb")) == 0 ) {
        perror ("fopen");
        return 3;
    }
    while ((c = fgetc (ifp)) != EOF)
        fputc (c, ofp);
    if (fclose (ifp) != 0 ) {
        perror ("fclose");
        return 4;
    }
    if (fclose (ofp) != 0 ) {
        perror ("fclose");
        return 5;
    }
    return 0;
}
```

* is repeated

Problem w/ fopen perror: which of the two fopen failed?

outputs as [fopen : (msg here)]

2510: Lecture 11 — February 3, 2017

→ Continued from Lecture 10 code...

↳ a program should check that it's being invoked correctly (validate cmd-line args)

↳ print a message about the CORRECT USAGE if the program is not invoked correctly

!!! minor problems with our program:

- if we succeed to open the first file but fail to open the second file, we did not explicity close the first file
- if fopen fails, the message doesn't tell us which of the 2 fopen failed:

FIX: print the name of the file as well

```
fprintf(stderr, "fopen %s: %s\n", argv[i], strerror(errno));
```

#include <errno.h>

▲ FILE POSITION INDICATOR

↳ there are 3 indicators associated with a stream:

// some operations automatically clear indicators *

① end of file indicator
② error indicator
③ file position indicator

} they can be cleared by using clearerr (refer to Lecture 9)

↳ if these indicators are set, most I/O operations will fail.

EOF → CTRL-D → clearerr

↳ indicates our current position within the file; by changing this, we can jump to another location

Seeking within a file

We'll look at 2 functions:

1) rewind — go back to beginning of file

```
rewind(fp);
```

2) fseek

```
fseek(fp, offset, whence);
```

FILE* — long — int (passed as whence value)

3 possible values:

1) SEEK_SET — relative to beginning of file
2) SEEK_CUR — current position
3) SEEK_END — end of file

ex. Seeking

kind of like rewind ←

```
fseek(fp, 0, SEEK_SET)   /* go to beginning of file */
fseek(fp, 0, SEEK_END)   /*    "     end of file    */
fseek(fp, 5, SEEK_CUR)   : move forward 5 bytes
fseek(fp, -5, SEEK_CUR)  : move back 5 bytes
```

* what

- most common OS allow seeking past the end of a file but they don't allow seeking past BEGINNING of a file

fseek returns -1 on failure:

could be -1 on failure or non-positive num on failure

```
if (fseek(fp, n, SEEK_CUR) != 0) { /* seek error */
    perror("fseek");
    /* additional error handling if necessary */
}
```

[NOT ALL streams support streams e.g. stdin, stdout, stderr]

Using ftell

↳ a lot of times, we want to seek back to a position that we remembered

stdin doesn't support ftell either

ftell - returns our current position

```
long pos = ftell(fp);
if (pos == -1) {        /* ftell failed */
    perror("ftell");
    /* additional error handling */
}
```

position can NOT be negative

BUFFERING

↳ stdin is typically line buffer
stdout "

NOTE: if you print a msg without a NEWLINE to stdout, it may not show up immediately.

↳ stderr is typically UNBUFFERED
- debugging msg should be printed to stderr

undefined behavior!

fflush(fp); only flushes output (DON'T USE W/ stdin)

(output stream or update stream)

[can you flush your toilet in?]