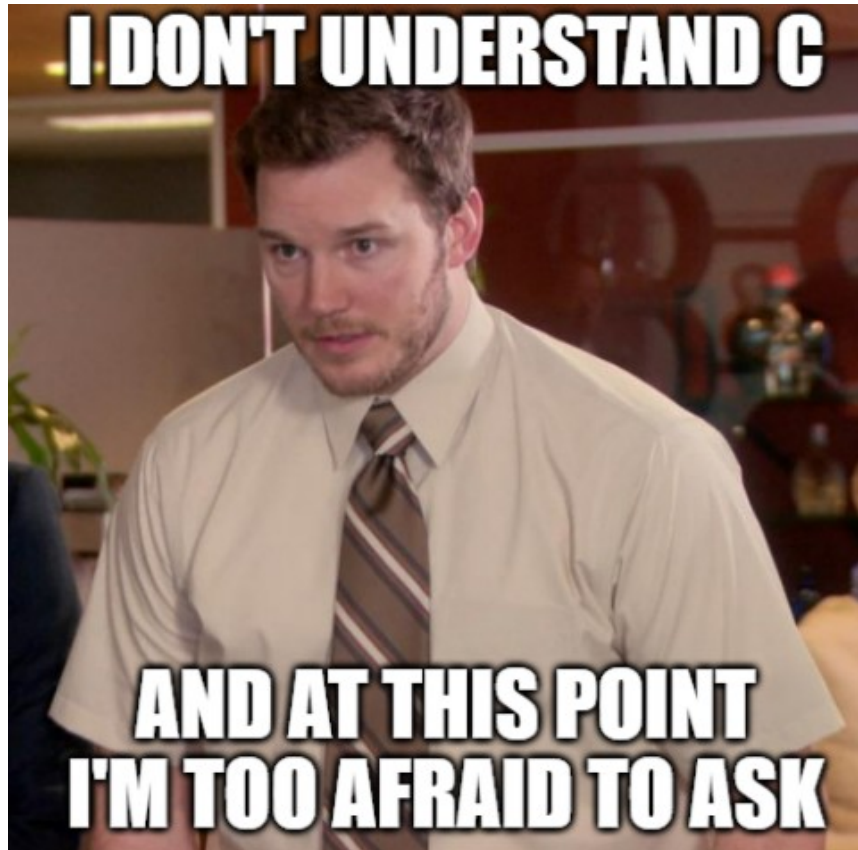# CSE 3100: Systems Programming

Lecture 9 : Input/Output and Files

Department of Computer Science and Engineering

University of Connecticut

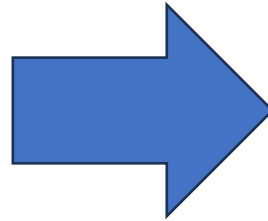Review of *Some* Previous Lecture Concepts…

Question 0: What will be printed?
A) r=7
B) r=0
C) r=1

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

int main(void){
    int x = 7;
    int y = 7;
    int r = (x == y);
    printf("r=%d\n", r);
    return 0;
}
```
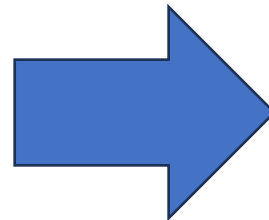
kaleel@CentralCompute:~$ ./test
r=1

Question 1: What will be printed?
A) x=10
B) x=7

```c
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int DoSomething(int x, int y){
5       int solution = x + y;
6       x = 10;
7       return solution;
8   }
9
10  int main(void){
11      int x = 7;
12      int y = 5;
13      int s = DoSomething(x, y);
14      printf("Value of x=%d\n", x);
15      return 0;
16  }
```

```
kaleel@CentralCompute:~$ ./test
Value of x=7
```
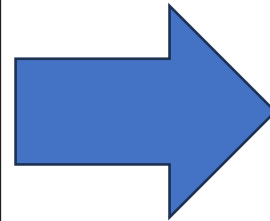
Why? Primitives are passed by VALUE to functions.

Question 2: What will be printed?
A) x=75
B) x=100

```c
4   typedef struct studentDataStruct{
5       int midtermGrade;
6       int homeworkGrade;
7   } sData; //Define a type for the structure
8
9   //Method to change grade
10  void changeHomeworkGrade(sData studentA){
11      studentA.homeworkGrade = 100; //wow so good
12  }
13
14  int main(void){
15      sData studentA;
16      studentA.homeworkGrade = 75;
17      changeHomeworkGrade(studentA);
18      printf("Value of x=%d\n", studentA.homeworkGrade);
19      return 0;
20  }
```

```
kaleel@CentralCompute:~$ ./test
Value of x=75
```

Why? Structures variables are passed by VALUE to functions.

Question 3: What will be printed?
A) x=75
B) x=100

```c
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   typedef struct studentDataStruct{
5       int midtermGrade;
6       int homeworkGrade;
7   } sData; //Define a type for the structure
8
9   //Method to change grade
10  void changeHomeworkGrade(sData* studentA){
11      (*studentA).homeworkGrade = 100; //wow so good
12  }
13
14  int main(void){
15      sData studentA;
16      studentA.homeworkGrade = 75;
17      sData* sPointer = &studentA;
18      changeHomeworkGrade(sPointer);
19      printf("Value of x=%d\n", studentA.homeworkGrade);
20      return 0;
21  }
```
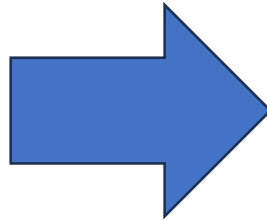
```
kaleel@CentralCompute:~$ ./test
Value of x=100
```

Why? The pointer to the structure allows us to manipulate the original variable.

Question 4: Are p0 and p1 the same?
A) Not same.
B) Same.

```c
1    #include <stdio.h>
2    #include <stdlib.h>
3
4    int main(void){
5        int* a = malloc(5*sizeof(int));
6        int* p0 = a;
7        int* p1 = &a[0];
8        printf("p0 = %p\n", p0);
9        printf("p0 = %p\n", p1);
10   }
```
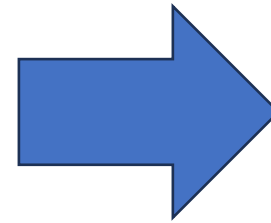
```
kaleel@CentralCompute:~$ ./test
p0 = 0x564b736d02a0
p0 = 0x564b736d02a0
```

Today's Lecture Topics

1. Error Handling

2. Files and Input/Output

1. Error Handling

errno

- Most C library functions can "fail"
  - When they do, they return a flag reporting failure... (-1)
  - Some set a global variable to report the exact error code

  errno

  // To use errno, include <errno.h>

- Check manual page to interpret the error code

- Print a more descriptive message with perror()

  ```
  void perror(const char *str);
  ```

Error Example

```
1    #include <stdio.h>
2    #include <errno.h>
3    #include <string.h>
4
5    //variable associated with error handling
6    extern int errno;
7
```

Code Example From: https://www.tutorialspoint.com/cprogramming/c_error_handling.htm

Error Example

```
1    #include <stdio.h>
2    #include <errno.h>
3    #include <string.h>
4
5    //variable associated with error handling
6    extern int errno;
7
8    int main () {
9
10       //pointer to a file
11       FILE * pf;
12       //try to open some file that doesn't exist
13       pf = fopen ("unexist.txt", "rb");
14
```

- Try to open some file that doesn't exist.
- We want to create an error on purpose to see the value of errno that is printed out.

Code Example From: https://www.tutorialspoint.com/cprogramming/c_error_handling.htm
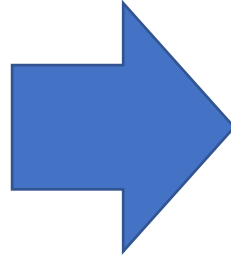
Error Example

```
15      if (pf == NULL) {
16          printf("Value of errno: %d\n", errno);
17      } else {
18          fclose (pf);
19      }
20      return 0;
21  }
```
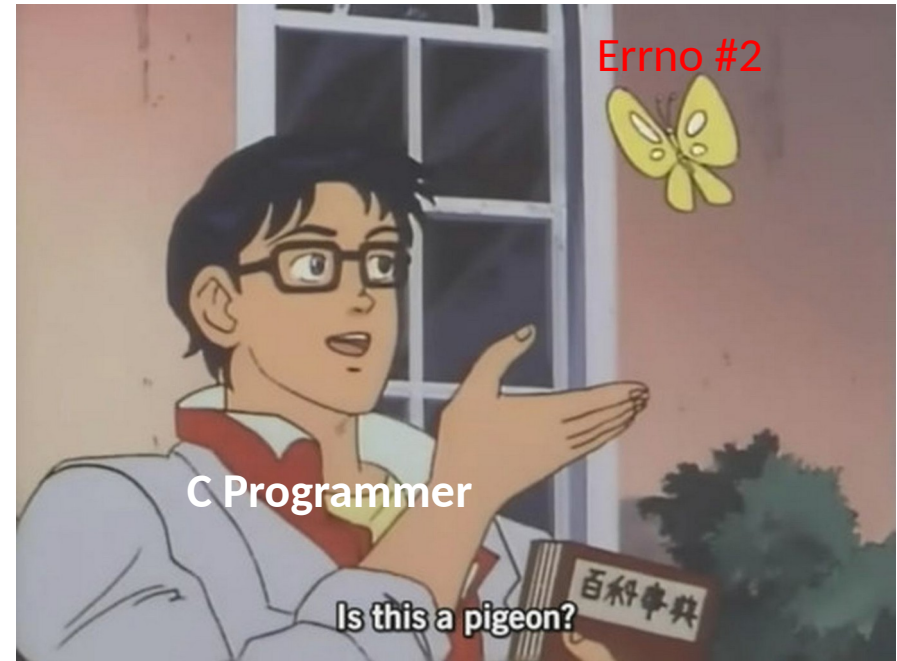
- If we don't have the file (NULL) then print out the value of errno.
- Otherwise just close the file.
- It is important to note: in this example we DO NOT have the file (on purpose).

Error Example

```c
1    #include <stdio.h>
2    #include <errno.h>
3    #include <string.h>
4
5    //variable associated with error handling
6    extern int errno;
7
8    int main () {
9
10       //pointer to a file
11       FILE * pf;
12       //try to open some file that doesn't exist
13       pf = fopen ("unexist.txt", "rb");
14
15       if (pf == NULL) {
16          printf("Value of errno: %d\n", errno);
17       } else {
18          fclose (pf);
19       }
20       return 0;
21    }
```

```
kaleel@CentralCompute:~$ ./test
Value of errno: 2
```

Errno #2

C Programmer

Is this a pigeon?

Code Example From: https://www.tutorialspoint.com/cprogramming/c_error_handling.htm

Error Example: What does no error look like?

```c
1   #include <stdio.h>
2   #include <errno.h>
3   #include <string.h>
4
5   //variable associated with error handling
6   extern int errno;
7
8   int main () {
9       //try not to have an error at all
10      printf("Value of errno: %d\n", errno);
11      return 0;
12  }
13
```
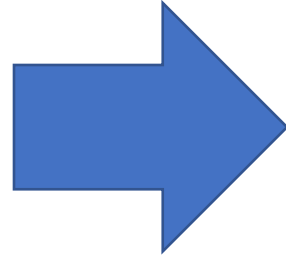
```
kaleel@CentralCompute:~$ ./test
Value of errno: 0
```

Ok so we know 0 means no error. But do we want to actually memorize all the C error codes?

Do we want to actually memorize all the C error codes?

- Yes. For the exam I will ask you to write all 131 error codes from your own memory (first 34 shown here):

ERROR CODE TABLE

Error number	Error Code	Error Description
1	EPERM	Operation not permitted
2	ENOENT	No such file or directory
3	ESRCH	No such process
4	EINTR	Interrupted system call
5	EIO	I/O error
6	ENXIO	No such device or address
7	E2BIG	Argument list too long
8	ENOEXEC	Exec format error
9	EBADF	Bad file number
10	ECHILD	No child processes
11	EAGAIN	Try again
12	ENOMEM	Out of memory
13	EACCES	Permission denied
14	EFAULT	Bad address
15	ENOTBLK	Block device required
16	EBUSY	Device or resource busy

17	EEXIST	File exists
18	EXDEV	Cross-device link
19	ENODEV	No such device
20	ENOTDIR	Not a directory
21	EISDIR	Is a directory
22	EINVAL	Invalid argument
23	ENFILE	File table overflow
24	EMFILE	Too many open files
25	ENOTTY	Not a typewriter
26	ETXTBSY	Text file busy
27	EFBIG	File too large
28	ENOSPC	No space left on device
29	ESPIPE	Illegal seek
30	EROFS	Read-only file system
31	EMLINK	Too many links
32	EPIPE	Broken pipe
33	EDOM	Math argument out of domain of func
34	ERANGE	Math result not representable

Just kidding!

A more readable error message:

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

//variable associated with error handling
extern int errno;

int main () {
    //pointer to a file
    FILE * pf;
    //try to open some file that doesn't exist
    pf = fopen ("unexist.txt", "rb");
    if (pf == NULL) {
        printf("Error opening file: %s\n", strerror(errno));
    } else {
        fclose (pf);
    }
    return 0;
}
```

```
kaleel@CentralCompute:~$ ./test
Error opening file: No such file or directory
```

2. Files and Input/Output

Files and Directories

- A file is an object that stores information, data, etc.

Example:

- Files you create with an editor (.c, .h, Makefile, readme, etc.)

- Executable generated by the compiler, and gcc itself

- Other devices, like screen, keyboard, …

- In Linux, files are organized in directories
  - A directory can have subdirectories and files
  - The top directory is /

- A path specifies the location of file/directory in the file system

    /home/john

- **In Unix/Linux, everything is a file**

The stdio library

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

- Declares FILE type and function prototypes
  - FILE is an **opaque type** (system dependent) for operating on files
    - It is a structure, but do not try to change it directly!
  - Use library functions to access FILE objects, via pointers (FILE *)
- Defines "standard" streams stdin, stdout, stderr ← They are FILE *
  - Created automatically when program starts
  - They are files!
- The library is linked automatically by the compiler

# Files and I/O API

- In C, a file is simply a sequential stream of bytes
- The "f" family of functions (fopen, fclose, fread, fgetc, fscanf, fprintf, …) are C library functions to operate on files
  - All these use a FILE* abstraction to represent a file
  - The C library provides buffering
    - That's why sometimes you do not see output of printf immediately

# File as stream of bytes

- Before use a file must be "open"
  - This sets a **position indicator** for reading and/or writing
- Each read/write starts from current position, and moves the indicator
  - Writing after last byte increases the file size
- Position indicator can also be changed with fseek
- All open files are closed when program ends
  - Good practice to close explicitly when no longer needed

offset: 0
Beginning of the file

file position indicator

offset: file size - 1
Last byte in the file
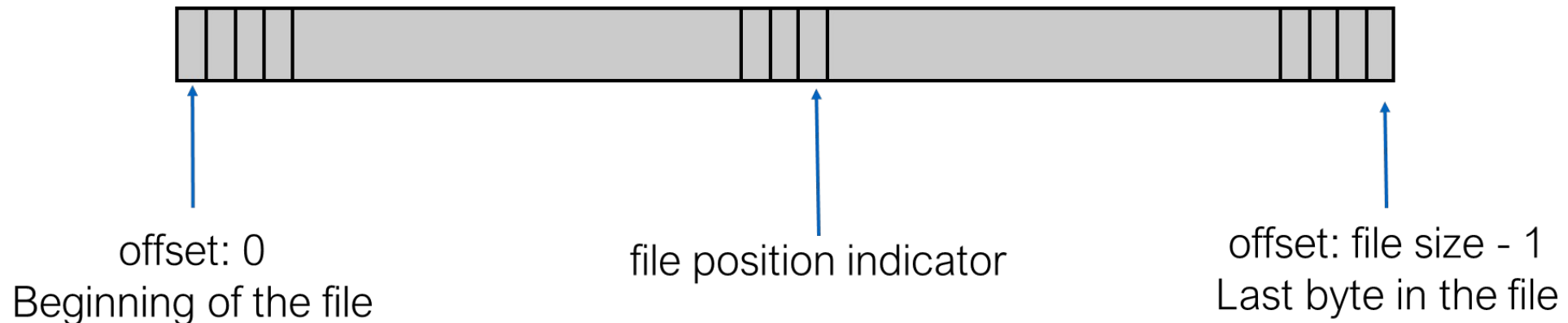
# Opening Streams

FILE* fopen( const char *filename, const char *mode);

- Open the file filename in mode as a stream of bytes
- Returns a pointer to FILE (FILE *) or NULL (and errno is set)
- Mode
    - "r"      : Reading mode
    - "r+"    : Read and write
    - "w"     : Writing mode, file is created or truncated to zero length
    - "w+"   : Read and write, but the file is created or truncated
    - "a"      : Append mode, the file is created if it does not exist
    - "a+"    : Read and append, the file is created if it does not exist.
                 Reading starts at beginning, but writing done at the end

# Closing Streams



a file

```
int fclose(FILE *stream);
```
- Close a stream
- Returns
  - 0               if it worked
  - EOF            if there was a problem (and errno is set)

# Simple File Reading Example

```c
1    #include <stdio.h>
2    #include <errno.h>
3    #include <string.h>
4    #define MAX_LINE_LENGTH 1000
5
6    extern int errno ;
7
8    int main () {
9        char    line[MAX_LINE_LENGTH];
10   //pointer to the file
11   FILE * pf;
12   pf = fopen ("HelloWorld.txt", "rb");
```

- Read in a maximum number of character lines from a text file.

What does the file "HelloWorld.txt" look like?

HelloWorld.txt

When you create a very complicated file:


```
Hello
World
My
Name
Is
Apple
```

Simple File Reading Example

```
12    pf = fopen ("HelloWorld.txt", "rb");
13    //make sure we can actually open the file
14    if (pf == NULL) {
15        printf("Error opening file: %s\n", strerror(errno));
16        return errno;
17    }
```

- Error handling as we previously discussed, make sure the file is readable.

Simple File Reading Example

```c
8    int main () {
9        char    line[MAX_LINE_LENGTH];
10   //pointer to the file
11   FILE * pf;
12   pf = fopen ("HelloWorld.txt", "rb");
```

- Create a file pointer and use fopen to point to the file.

```c
18   while(fgets(line, MAX_LINE_LENGTH, pf)){
19       printf(line);
20   }
```

- Use fgets to keep reading lines of the file until we reach the end or max length.

Who is fgets?

`char *fgets(char *str, int n, FILE *stream)`

- The C library function **char *fgets(char *str, int n, FILE *stream)** reads a line from the specified stream and stores it into the string pointed to by **str**.

- It stops when either **(n-1)** characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.

Simple File Reading Example

```c
1   #include <stdio.h>
2   #include <errno.h>
3   #include <string.h>
4   #define MAX_LINE_LENGTH 1000
5
6   extern int errno ;
7
8   int main () {
9       char    line[MAX_LINE_LENGTH];
10      //pointer to the file
11      FILE * pf;
12      pf = fopen ("HelloWorld.txt", "rb");
13      //make sure we can actually open the file
14      if (pf == NULL) {
15          printf("Error opening file: %s\n", strerror(errno));
16          return errno;
17      }
18      while(fgets(line, MAX_LINE_LENGTH, pf)){
19          printf(line);
20      }
21      fclose(pf);
22      return 0;
23  }
```
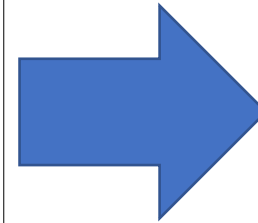
```
kaleel@CentralCompute:~$ ./test
Hello
World
My
Name
Is
Applekaleel@CentralCompute:~$
```

Byte Sized Reading

- In the previous coding example we looked at reading an entire line.

- What if we want to read bytes at a time?

```
int fgetc( FILE *stream);
int fputc(int c, FILE *stream);
```

- Read or write one (ASCII) character (8-bits) at a time
  - Can be slow for large files
- fgetc reads a character from the stream and returns the character just read in (as unsigned char extended to int)
  - Returns EOF when at the end of file or on error
- fputc writes the character received as argument to the stream and returns the character that was just written out
  - Returns EOF on error

getc / putc and ungetc

```
int getc(FILE *stream);
int putc(int c, FILE *stream);
```
- Same as fgetc/fputc except they may be implemented as macros
- Use fgetc/fputc unless you have strong reasons not to

```
int ungetc(int c, FILE *stream);
```
- Pushes last read char back to stream, where it is available for subsequent read operations
- Only one pushback guaranteed

getchar / putchar

```
int getchar(void)
// same as fgetc(stdin)
```
- Reads a character from stdin
- Returns the character just read in, or EOF on end-of-file or errors

```
int putchar(int c)
// same as fputc(c, stdout)
```
- Writes the character received as argument on stdout
- Returns the character that was just written out, or EOF on errors

Formatted output

```
int fscanf(FILE *stream, const char
*format, ...);
int fprintf(FILE *stream, const char
*format, ...);
```

- Formatted input from file and output to file
- Like scanf()/printf(), but not from stdin or to stdout

Fprintf Example

```
1    #include <stdio.h>
2    #include <errno.h>
3    #include <string.h>
4    #define MAX_LINE_LENGTH 1000
5
6    extern int errno ;
7
8    int main () {
9        char    line[MAX_LINE_LENGTH];
10       //pointer to the file
11       FILE * pf;
12       pf = fopen ("HelloWorld.txt", "r+");
13       //make sure we can actually open the file
14       if (pf == NULL) {
15           printf("Error opening file: %s\n", strerror(errno));
16           return errno;
17       }
18       while(fgets(line, MAX_LINE_LENGTH, pf)){
19           //printf(line);
20       }
21       fprintf(pf, "\nOrange");
22       fclose(pf);
23       return 0;
24   }
```

HelloWorld.txt

Hello
World
My
Name
Is
Apple

BEFORE the code is run.

HelloWorld.txt

Hello
World
My
Name
Is
Apple
Orange

AFTER the code is run.

Moving file position indicator

```
long ftell(FILE *stream);
```
- Read file position indicator
- Return -1 on error

```
int fseek(FILE * stream, off_t offset, int whence);
```
- Set the file position indicator
- Return 0 on success and -1 on error

Example:

```
fseek(fp,   0, SEEK_SET); // move to the beginning
fseek(fp, 200, SEEK_CUR); // move forward 200 bytes
fseek(fp,  -1, SEEK_END); // move to the last byte
```

Code Example: Finding the size of a file

```c
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main() {
5       FILE * pf;
6       double    numbytes;
7       pf = fopen("HelloWorld.txt", "r");
8       //Use fseek to go to the end of the file
9       fseek(pf, 0L, SEEK_END);
10      //read the file position indicator
11      numbytes = ftell(pf);
12      printf("Number of bytes in file:%f\n", numbytes);
13      return 0;
14  }
```
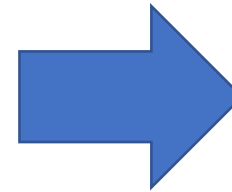
```
kaleel@CentralCompute:~$ ./test
Number of bytes in file:35.000000
```

Source: https://www.tutorialkart.com/c-programming/c-read-text-file/

Related I/O Function Slides

```
//Check if end-of-file is set (after a read attempt!)
int feof(FILE * stream);

//Force write of buffered data
int fflush(FILE * stream);


int fputs(const char *str, FILE *out)
```
- Writes the string str to out, stopping at '\0'
- Returns number of characters written or EOF

```
size_t fread (void *ptr, size_t sz, size_t n, FILE *stream);
size_t fwrite(void *ptr, size_t sz, size_t n, FILE *stream);
```
- Read / write a sequence of byte from/to a stream
- Return the number of items read or written
  - If smaller than n, EOF or error

Figure Sources

1. https://cdn.vox-cdn.com/thumbor/VKx8R0U9BKwV9SE2_eaVwJtywjU=/0x0:500x375/1400x1050/filters:focal(158x110:238x190):format(jpeg)/cdn.vox-cdn.com/uploads/chorus_image/image/59741997/n4scgse21iuz.0.jpg

2. https://www.thegeekstuff.com/2010/10/linux-error-codes/

3. https://i.kym-cdn.com/photos/images/facebook/001/100/963/ab8.jpg

4. http://www.quickmeme.com/meme/3upt4q

5. https://live.staticflickr.com/699/32421179826_69211d9189_b.jpg

6. https://external-preview.redd.it/6oAehEdLMEkhf2vzvah