File Input and Output

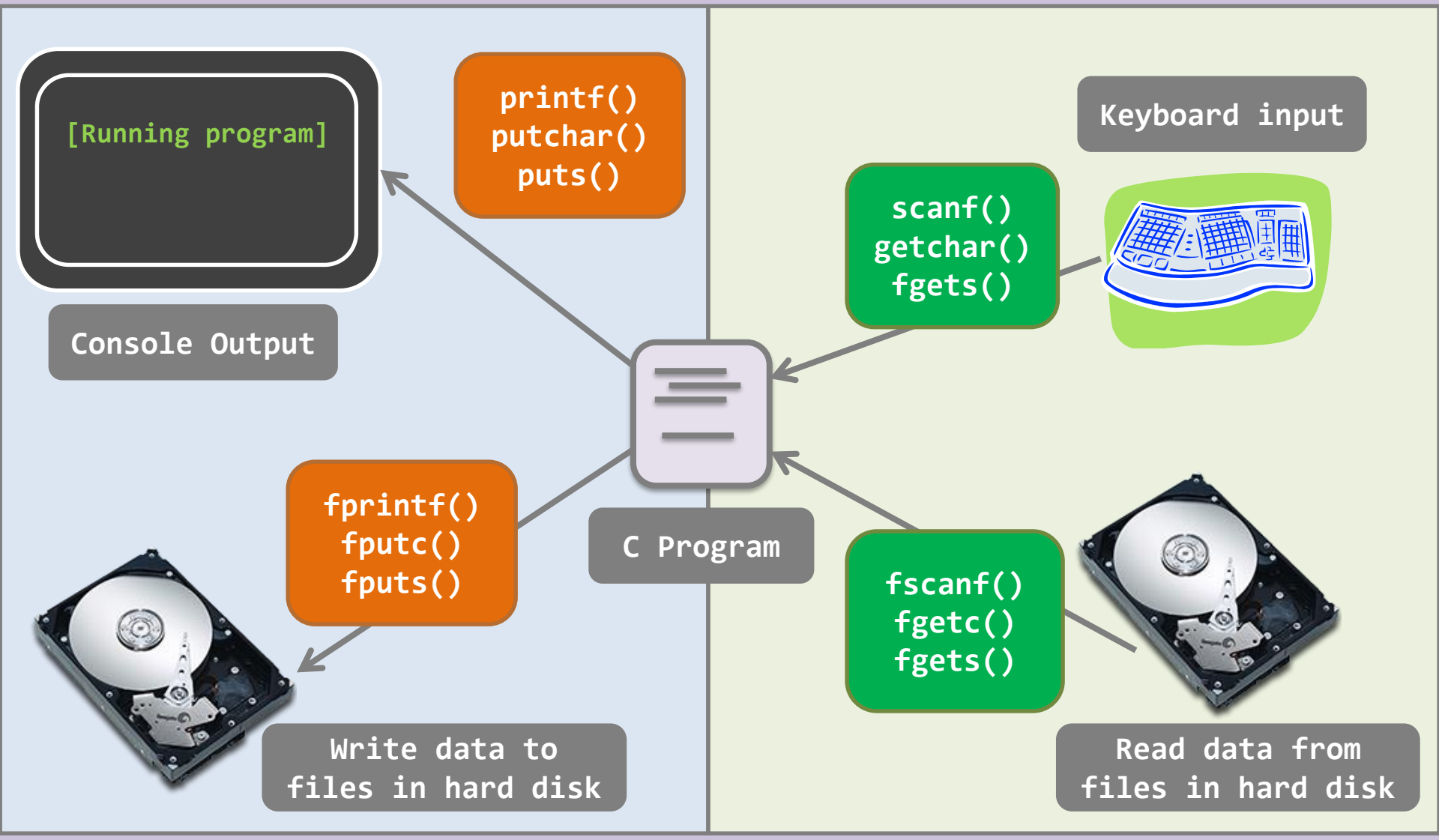
(exam included)

Outline

- Introduction + Read and Write data
  - Numbers
  - Characters
  - Strings
- Fun Examples:
  - Count file size
  - Copy files
  - Caesar Cipher

Streaming concept

File I/O – Big picture



Output stream **Input stream**

File I/O – File reading

- **Data reading**: reading data from a file is similar to reading data from the keyboard

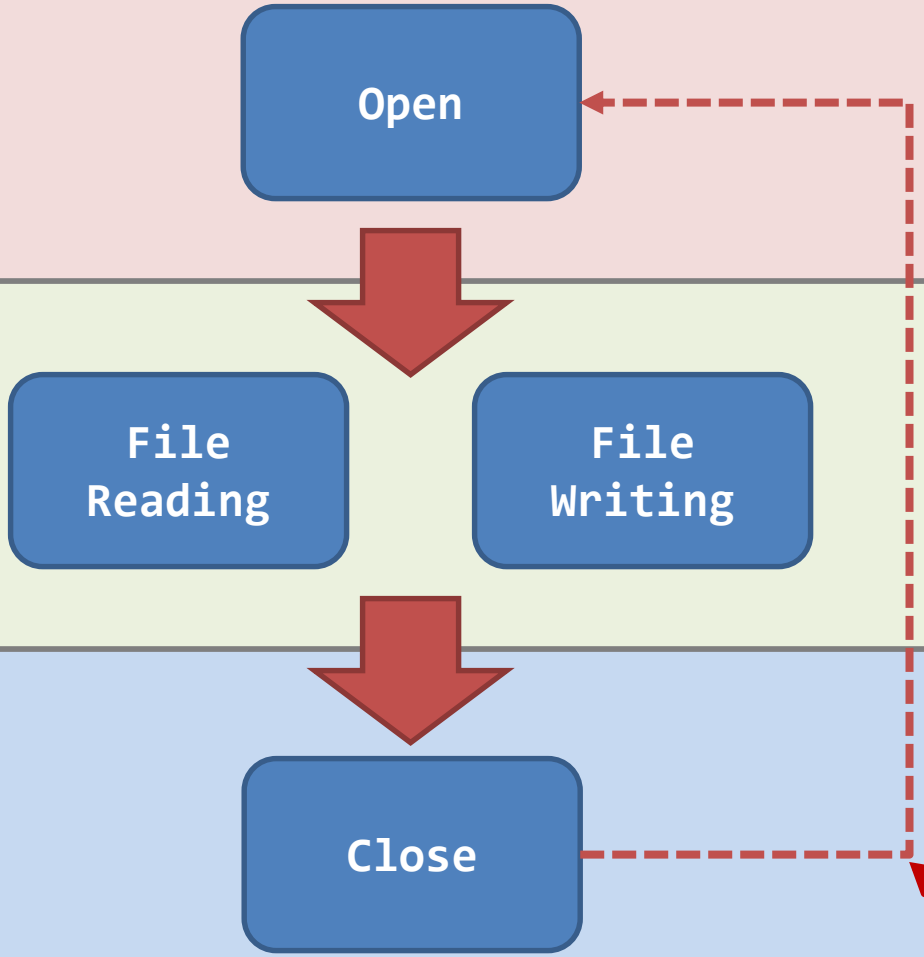
- **Keyboard input** VS **File input**
  - **Similarity**: data read in a FIFO (first-in-first-out) manner
  - **Keyboard input**: the program *needs to wait* for the input until the input is typed in by the user
  - **File input**: data is *readily available*

File I/O – File writing

- **Data writing**: writing data to a file is similar to writing data (messages) to the screen, or the console output

- **Console Output** VS **File Output**
  - **Similarity**:
    - Data written in a FIFO (first-in-first-out) manner
    - Data written to the screen / file in a line-by-line manner

File I/O – Flow / Procedure



Opening a file involves 2 parameters:
- Filename, and
- Opening mode;

Reading and Writing include data:
- Numbers,
- Characters and Strings, etc.

Once an opened file is closed, your program declares that the file is not needed from that moment.

Need to **re-open the file** if you need to process the file again.

Example #1: Read data from a file

```
1   #include <stdio.h>
2
3   int main( void )
4   {
5       FILE * fp ;
6       int    num1 , num2 ;
7
8       fp = fopen( "data.txt" , "r" );   // open file for ‘r’eading
9
10      // Read two integers from the file
11      fscanf( fp , "%d%d"     , & num1 , & num2 );
12      printf(      "%d %d\n" ,   num1 ,   num2 );
13
14      fclose( fp );  // close file
15      return 0 ;
16  }
```

Content of "data.txt"

```
123
456
```

Console Output

```
123 456
```

Step 1: Open a file by fopen()

```
FILE * fp = fopen( "data.txt" , "r" );
```

Data type **FILE** is called a "file structure"

"**FILE ***" means **a pointer to the file structure (or file pointer)**. See ***** in pointer lecture notes.

fopen() is the function to open a file:

- **1st argument**: the location of a file in the computer file system (directory), e.g., "data.txt" means the file is **in the same folder as the program**.
- **2nd argument**: it is a string that defines the **file opening mode**, e.g., "r" means opening the file in "**read-only**" mode.

Note: windows (older versions) use back slash while others use forward slash for folders: https://en.wikipedia.org/wiki/Home_directory

Open a file: more on opening modes

Mode	Description
"r"	**Read only**. - The target file must exist - Cannot read from a write-only file E.g., `FILE * fp = fopen( "data.txt" , "r" );`
"w"	**Write only**. - If the target file does not exist, such a file will be created - If the target file exists, **all existing data in the file will first be erased**, so the file size becomes zero. Then, we will write to the empty file - Cannot write to a read-only file E.g., `FILE * fp = fopen( "data.txt" , "w" );`
"r+"	Behave like "r", but allow writing to the file
"w+"	Behave like "w", but allow reading from the file

* Note: other file opening modes such as "a" for append; http://www.cplusplus.com/reference/cstdio/fopen/

Open a file: return value of fopen()

- If **fopen()** fails, it will return NULL;
- Else, the return value is not NULL.
  - Use **exit() function** to stop the program immediately

```
1   #include <stdio.h>
2   #include <stdlib.h>  // required by exit()
3   int main( void )
4   {
5       char filename[] = "data.txt" ;
6       FILE * fp = fopen( filename, "r" );
7       if ( fp == NULL )
8       {
9           printf( "Error: cannot open file [%s]!\n" , filename );
10          exit( 1 );  // exit(1) - for unexpected termination
11      }               // exit(0) – for expected termination
12      return 0 ;
13  }
```

Step 2: Read data from file by `fscanf()`

- **`fscanf()`** is the file version of **`scanf()`**

```
scanf( "%d%d" , & num1 , & num2 );
```

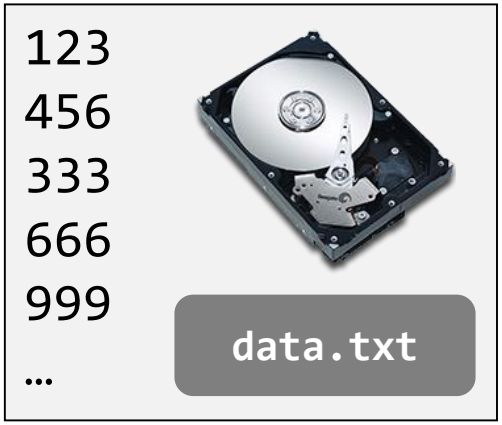
VS

```
fscanf( fp , "%d%d" , & num1 , & num2 );
```

The data type here should be **`FILE *`**, which corresponds to the return value from a successful call to `fopen()`.

Example: Read integers from a file

- Goal:
  - To read the data from a file in the format below, i.e., each line contains one integer
    - We don't know how many numbers inside the file
    - We can decide by using the return value of **`fscanf()`**
  - Compute the sum of all the integers read and print the result to the screen



Example: Read integers from a file

```
1   int main( void )
2   {
3       FILE * fp ;
4       int     num , sum = 0 ;
5
6       fp = fopen( "data.txt" , "r" );  // open file (omit check: NULL)
7       while ( 1 )
8       {
9           if ( fscanf( fp , "%d" , & num ) != 1 )  // one data value
10              break ;
11          sum += num ;
12      }
13      fclose( fp ) ;  // close file
14      printf( "Sum = %d\n" , sum );
15
16      return 0 ;
17  }
```

Remember the meaning of the return value of **`scanf()`**?

Answer:

The number of items read.

So does **`fscanf()`**!

Example: Read numbers from a file

- Goal:
  - To read the data from a file in the format below:
    - Each line has 1 integer followed by 2 floating-point numbers
  - Compute the sum of all the numbers read and print the result to the screen

```
123    1.0    2.0
456    3.5    4.4
333    2.6    7.8
666    7.7    3.32
999    9.2    5.2
…
```

data.txt

Example: Read numbers from a file

```
1  int main( void )
2  {
3     FILE * fp   ;
4     int    num1 ;
5     double num2 , num3 , sum = 0 ;
6
7     fp = fopen( "data.txt" , "r" );  // open file (omit check: NULL)
8     while ( 1 )
9     {
10       if ( fscanf( fp , "%d%lf%lf" , &num1 , &num2 , &num3 ) != 3 )
11          break ;
12       sum += num1 + num2 + num3 ;
13    }
14    fclose( fp ) ;  // close file
15    printf( "sum = %f\n" , sum );
16    return 0 ;
17 }
```

Read data from a file: other functions

Function	Description and Example
`fgetc()` Read one character from a file	File version of **`getchar()`** - **Read one character** from the opened file - **Return EOF** when reaching the end of the file Example:

```
...
   char   input ;
   FILE * fp = fopen( "data.txt" , "r" );
   while ( 1 )
   {
       input = fgetc( fp );
       if ( input == EOF )
           break ;
   }
   fclose( fp );
...
```

Read data from a file: other functions

Function	Description and Example
`fgets()` Read a line from a file	To use `fgets()` with keyboard, we write: - **`fgets( string , size_of_string , stdin );`** For file input, we change **`stdin`** to the file structure, see below. Like the keyboard case, **`fgets()`** stores the **<u>trailing newline '\n' character</u>** in the given character array (see `string` below).

```
...
   char   string[ 128 ];
   FILE * fp = fopen( "data.txt" , "r" );
   while ( 1 )
   {
      if ( fgets( string , 100 , fp ) == NULL )
         break ;
   }
   fclose( fp );
...
```

Step 2: Write data to a file by `fprintf()`

- **`fprintf()`** is the file version of **`printf()`**

```
printf( "%d\n%d\n" , num1 , num2 );
```

Console Output

VS

```
fprintf( fp , "%d\n%d\n" , num1 , num2 );
```

```
123
456
```

data.txt

The data type here should be **`FILE *`**, which corresponds to the return value from a successful call to `fopen()`.

Example: Write data to a file

```c
#include <stdio.h>

int main( void )
{
    FILE * fp ;

    fp = fopen( "data.txt" , "w" );

    fprintf( fp , "Hello!\n" );
    for ( int i = 0 ; i < 10 ; i++ )
        fprintf( fp , "%d " , i );
    fprintf( fp , "\n" );

    fclose( fp );

    return 0 ;
}
```

Open a file as write-only

All data inside "data.txt" will first be erased (if it exists)

As a matter of fact, we produce a file with **plain-text content**

data.txt

```
Hello!
0 1 2 3 4 5 6 7 8 9 
```

Write data to a file: other functions

Function	Description and Example
`fputc()` Write one character to a file	File version of **`putchar()`** - **<u>Print one character</u>** to the opened file - **<u>Return EOF</u>** when encountering any kind of errors, e.g., disk full, write to a read-only (opened) file, etc. Example:

```c
int main( void )
{
    FILE * fp = fopen( "data.txt" , "w" );
    for ( int i = 0 ; i < 10 ; i++ )
        fputc( i + '0' , fp );
    fclose( fp ) ;
    return 0 ;
}
```

```
0123456789
```

data.txt

Write data to a file: other functions

Function	Description and Example
`fputs()` Write a string to the file	The function writes a string to the opened file, with an **extra newline character** appended to the file Example:

```c
int main( void )
{
    char string[ 100 ] = "hello" ;
    FILE * fp = fopen( "data.txt" , "w" );
    fputs( string , fp );
    fputs( string , fp );
    fclose( fp );
    return 0 ;
}
```

```
hello
hello
```

data.txt

Last but not least: `fclose()`

- **`fclose()`** tells the Operating System (OS) to release the resource allocated for opening the file.
  - It is a good practice to close each opened file
  - Because every OS imposes **<u>a limit on the maximum number of files that a running program can open</u>**., etc.

```
int main( void )
{
   FILE * fp = fopen( "data.txt" , "w" );
   for ( int i = 0 ; i < 10 ; i++ )
      fputc( i + '0' , fp );
   return 0 ;  // not closing file
}
```

Bad Practice!

* Personal habit: after writing fopen, I will write fclose in the next line as a reminder

Outline

- ~~Introduction + Read and Write data~~
  - ~~Numbers~~
  - ~~Characters~~
  - ~~Strings~~
- Fun Examples:
  - Count file size
  - Copy files
  - Caesar Cipher

Fun example #1: Count file size

- Goal:
  - Target file: "data.txt"
  - Print the number of bytes of "data.txt" to the screen

- Discussion:
  - What file opening mode we need?
  - Use which file I/O function?

Fun example #1: Count file size

```c
int main( void )
{
    FILE * fptr ;
    int    count = 0 ;

    fptr = fopen( "data.txt" , "r" ); // read-only (omit check NULL)
    while ( 1 )
    {
        if ( fgetc( fptr ) == EOF )
            break ;
        count++ ;
    }
    fclose( fptr );   // close file
    printf( "number of bytes = %d\n" , count );

    return 0 ;
}
```

Can we simplify the code?

Fun example #1: Count file size (simplified)

```c
int main( void )
{
    FILE * fptr ;
    int    count = 0 ;

    fptr = fopen( "data.txt" , "r" ); // read-only (omit check NULL)

    while ( fgetc( fptr ) != EOF )
        count++ ;

    fclose( fptr );   // close file
    printf( "number of bytes = %d\n" , count );

    return 0 ;
}
```

Fun example #2: Copy files

- Goal:
  - Copy the contents in "**`input.txt`**" to "**`output.txt`**"

- Discussion:
  - Open how many files?
  - File opening modes?
  - Use which functions?

Fun example #2: Copy files

```
1   int main( void )
2   {
3       FILE * fptr_in  ;    // input file
4       FILE * fptr_out ;    // output file
5       int    c    ;
6
7       fptr_in  = fopen( "input.txt"  , "r" );    // read-only
8       fptr_out = fopen( "output.txt" , "w" );    // write-only
9       while ( 1 )
10      {
11          c = fgetc( fptr_in );
12          if ( c == EOF )
13              break ;
14          fputc( c , fptr_out );
15      }
16      fclose( fptr_in  );
17      fclose( fptr_out );    // close both files
18      return 0 ;
19  }
```

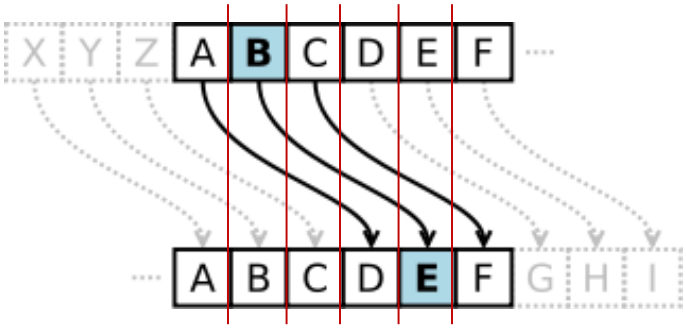
Q: Any other way?

A: we may also copy line by line

Fun example #3: Caesar Cipher

- What is it? Let's see
  https://www.youtube.com/watch?v=sMOZf4GN3oc

- The encoder:
  – Shift all characters **forward by 3**



Source: wikipedia

- The decoder:
  – Shift all characters **backward by 3**

Fun example #3: Caesar Cipher

- The Codec (coder-encoder):
  - Two files: "**`input.txt`**" and "**`output.txt`**"
  - File opening modes?
  - Use which file reading/writing functions?
  - Logic of character shifting?

<u>No codes will be given in this set of slides</u>

Please treat this as a self-test exercise

Hint: Treat the file copier code as the base of writing both the encoder and the decoder. Furthermore, you may try the XOR idea you learnt in bitwise operator and write an encoder and a decoder.

Miscellaneous Topics

~~ Standard I/O streams ~~

Standard I/O streams

- By the way, what exactly is the difference between **`printf()`** and **`fprintf()`**?

`printf(`	`[format string],`	`[list of parameters] )`
`fprintf( [opened file],`	`[format string],`	`[list of parameters] )`

  – They look like twins!!

Standard I/O streams

- Yes, **`printf()`** and **`fprintf()`** are twins!

This is a definition in **`stdio.h`**:
- `FILE * stdout ;`

`stdout` is called the
standard output stream

`stdout` represents the screen output. When you write to **`stdout`**, you actually write to the screen.

```
printf( "%d\n" , 123 );
```

SAME

```
fprintf( stdout , "%d\n" , 123 );
```

Standard I/O streams

- The three streams are defined in **`<stdio.h>`**
  - That's why it is called "**standard I/O header file**"

	Variable name type: `FILE *`	Description
Standard Input Stream	**`stdin`**	Represents the keyboard input
Standard Output Stream	**`stdout`**	Represents the screen output Usually for printing normal output
Standard Error Stream	**`stderr`**	Represents the screen output Usually for printing error messages

Standard I/O streams

- Twins examples (identical pairs of functions):

Reading from keyboard	Printing to Screen
`scanf( "%d" , & number );`	`printf( "%d" , number );`
`fscanf( stdin , "%d" , & number );`	`fprintf( stdout , "%d" , & number );`
`int ch = getchar() ;`	`putchar( ch );`
`int ch = fgetc( stdin );`	`fputs( stdout , ch );`

Other useful File I/O functions

- E.g.,
  - `feof()`: check if we reached the end-of-file indicator
  - `ftell()`: get the current position in a stream (e.g., file)
  - `fseek()`: reposition a stream's (file) position
  - `rewind()`: set the current position to the beginning
  - `tmpfile()`: open a temporary file
  - fflush(): flush unwritten data in output buffer to file

See http://www.cplusplus.com/reference/cstdio/