Short-term Hands-on Supplementary Course on C Programming

SESSION 12: File Handling

KARTHIK D
NIVEDHITHA D

Mode: **Asynchronous**
Location: Online

Agenda

1. File Handling — What and Why
2. File Data Formats
3. File Descriptors, Streams and Modes
4. Handling Text Files in C
5. Handling Binary Files in C
6. File positioning — seek, tell
7. Tutorial: Random Access in Files
8. Next Session: Structures and Files

Administrative Instructions

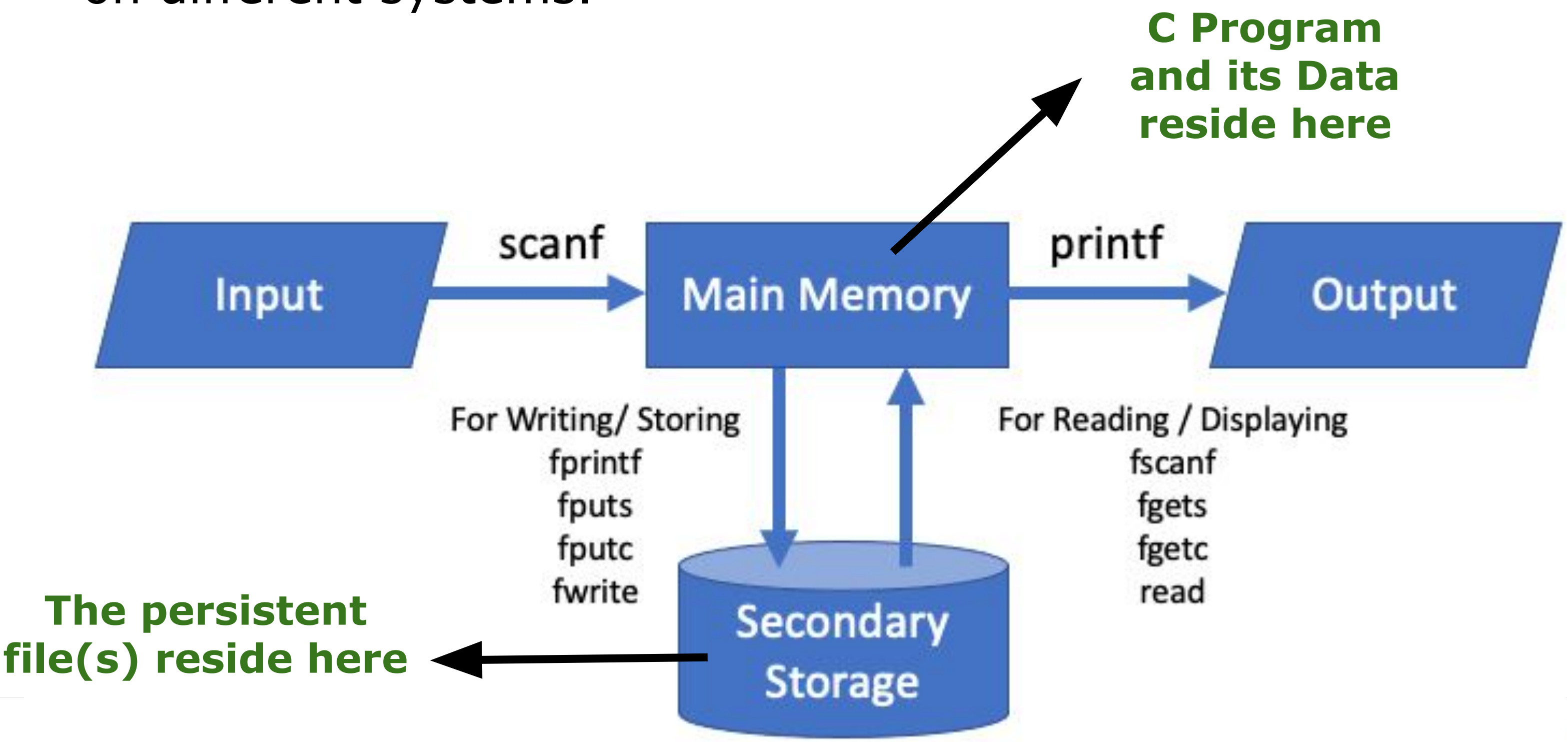
- Please fill out the feedback form - will be shared in the chat
- Join us on Microsoft Teams,

  Team Code: **rzlaicv**

GITHUB REPOSITORY! ⭐🌟**

File Handling

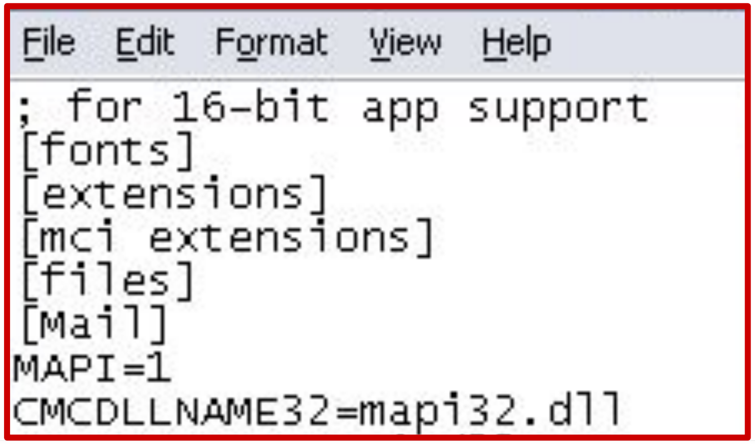
- To **persistently preserve** your data even after program termination.
- To **read large amount of data** — access them from a file using C functions.
- To **move data** from one computer to another, if your program runs on different systems.



File Data Formats

Text Files

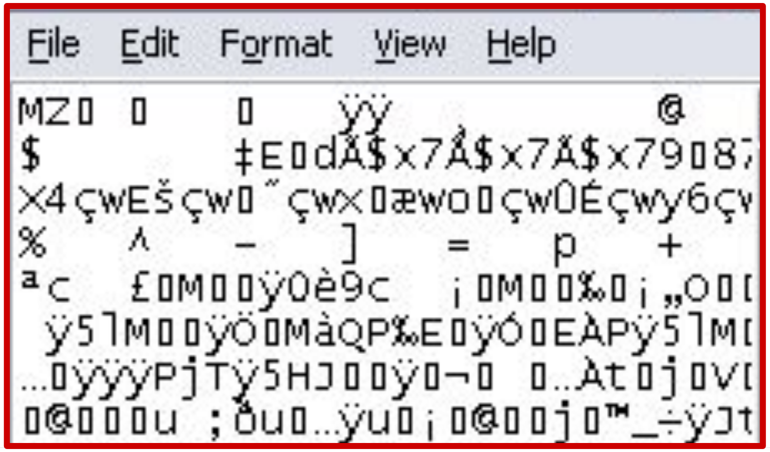
- The bits represent **text characters,** usually in the ASCII format
- Can store only “**plain-text**” data
- **Human-readable**, accessible through any text editor
- **Less secure** — can be edited by anyone



```
; for 16-bit app support
[fonts]
[extensions]
[mci extensions]
[files]
[Mail]
MAPI=1
CMCDLLNAME32=mapi32.dll
```

Binary Files

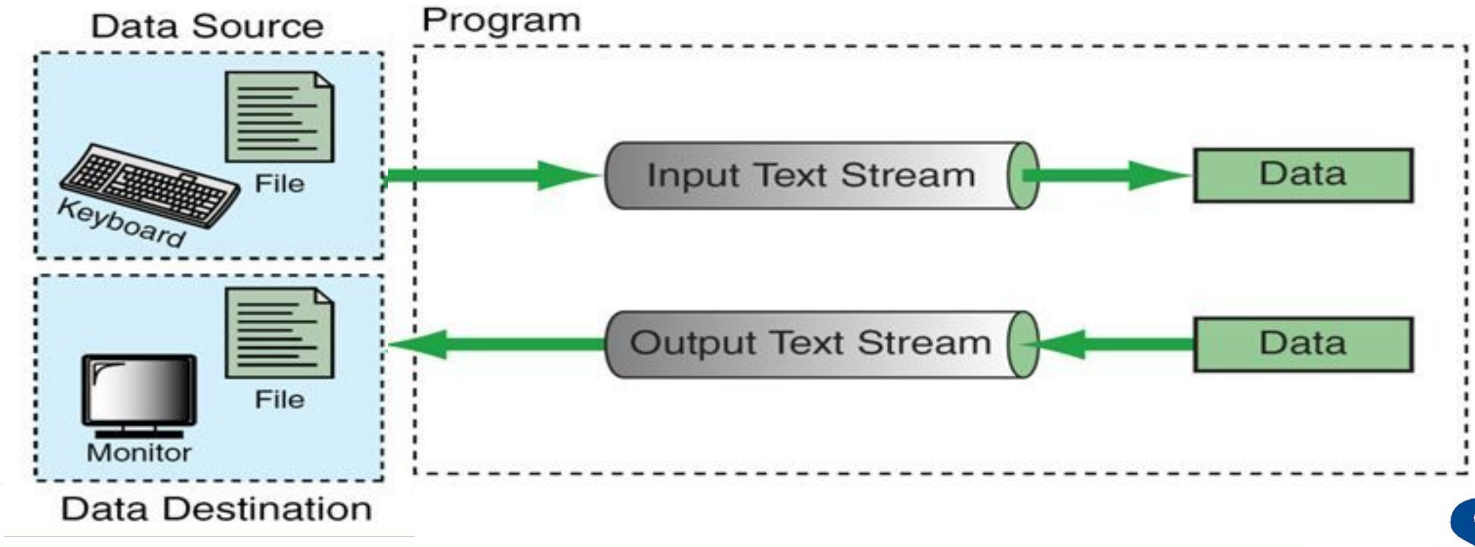
- The bits represent **custom data** — **program memory** is dumped into the file
- Can store **any form of data** — audio, image, etc.
- Readable only using the data format it was written with
- **More secure** — not easy to edit meaningfully



```
MZ□  □     □    ÿÿ              @
$          ‡E□dÅ$x7Å$x7Å$x79□87
X4çwEšçw□˜çwx□æwo□çw0Éçwy6çv
%   ^    –    ]    =    p    +
ªc   £□M□□ÿ0è9c   ¡□M□□‰□¡„O□[
 ÿ5]M□□ÿÖ□MàQP‰E□ÿÓ□EÀPÿ5]M[
…□ÿÿÿPjTÿ5HJ□□ÿ□¬□  □…At□j□V[
□@□□□u ;ôu□…ÿu□¡□@□□j□™_÷ÿJt
```

File Streams and Descriptors

- To operate on a file, we first set up a “**connection**” between the program and the file
- There are broadly 2 **file connection representation mechanisms**,
  - **File descriptors** (*int*): Low-level, primitive
  - **File streams** (*FILE**): High-level, layered atop descriptors
- To keep things simple, <u>we use streams in this course</u>
  - Streams offer more **operability on I/O buffering**
  - Streams provide **simpler, richer** and **powerful I/O functions**



File Streams and Descriptors

DO YOU REALIZE? The **printf()** and **scanf()** functions we've been using all along, essentially operate on standard I/O streams that connect to **STDIN** (Standard I/P) and **STDOUT** (Standard O/P)

- The functions we'll use to handle file I/O are all defined in the **<stdio.h> header** file

- To **establish a stream connection** with a file, we use the *fopen()* function,

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

  [Reference]

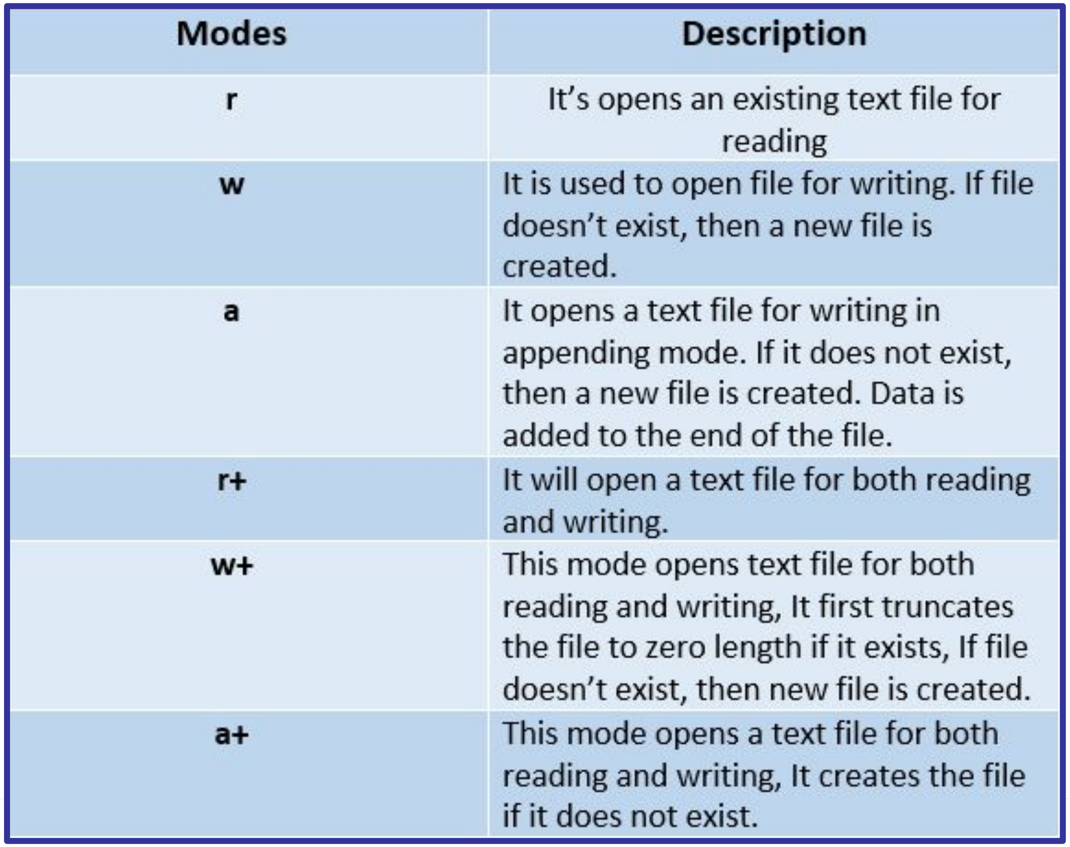
- To **terminate a stream connection** with a file, we use the *fopen()* function,

  int fclose (**FILE *** stream_object);

  [Reference]

File Access Modes

- File streams offer granularity to specify "**how**" you want to **open (connect to) a file**
  - For instance, if a file with the same name exists, should *fopen()* replace it with a new file, or append to it?
- This is specified using the **mode argument** in ***fopen()***



Modes	Description
r	It's opens an existing text file for reading
w	It is used to open file for writing. If file doesn't exist, then a new file is created.
a	It opens a text file for writing in appending mode. If it does not exist, then a new file is created. Data is added to the end of the file.
r+	It will open a text file for both reading and writing.
w+	This mode opens text file for both reading and writing, It first truncates the file to zero length if it exists, If file doesn't exist, then new file is created.
a+	This mode opens a text file for both reading and writing, It creates the file if it does not exist.

Add a "b" to the mode string, to open the file in **binary mode**.
Eg: r → rb

A **complete list** of allowed **file mode strings** can be found here

File Operations

- In C, the **typical operations** on files, either *text* or *binary*, include:
    - Creating a new file
    - Opening an existing file
    - Reading data from a file
    - Writing data to a file
    - Closing a file
- The processing, formatting, etc. go into program logic

- We'll now perform some basic I/O operations on *text* and *binary* files

Text File I/O

Once we have the *file stream object* — **stream**,

- To write a string to a file,

  size_t fprintf (**FILE *** stream**, const char *** format**,** ...);

  [Reference]

- To read a string from a file,

  size_t fscanf (**FILE *** stream**, const char *** format**,** ...);

  [Reference]

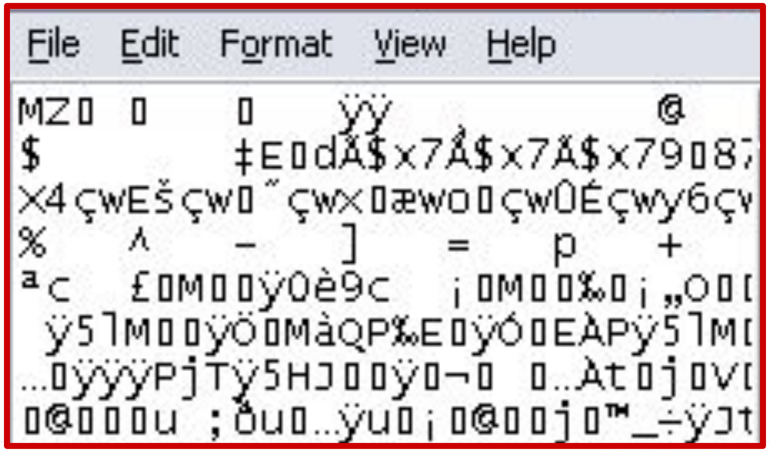
NOTE: Again, just a variation of the **printf()** and **scanf()** functions — just choose a file stream instead of standard I/O

NOTE: A whole range of other I/O functions are available, often specific to certain formats of data. Here's a list!

Binary Files in C [ref]

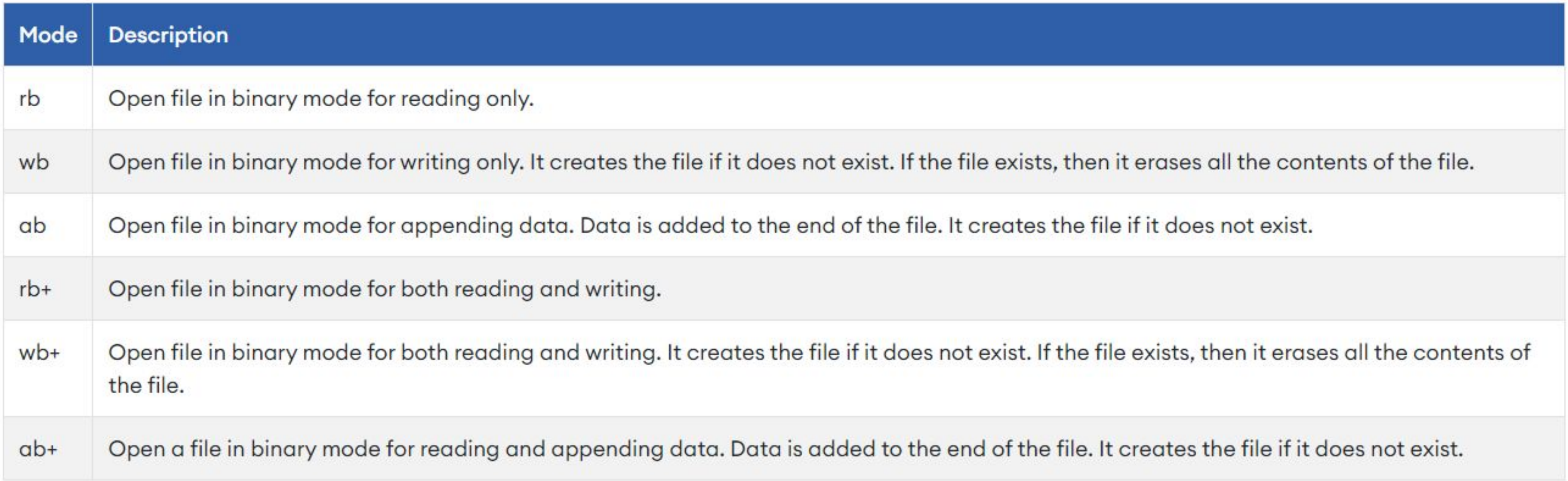
- The bits represent **custom data** — **program memory** is dumped into the file **as a sequence of bytes** (compiled text file)
- Can store **any form of data** — audio, image, etc
- Readable only using the data format it was written with (.java → .class)
- **More secure** — not easy to edit meaningfully
- You can jump instantly to position in the file (**random access** for structure-wise parsing based on size)

- **Hard to eliminate errors** in bin files as compared to text file
- Handling of **newlines** and storage of **numbers** is efficient
- **Absence of EOF** - size based



```
MZ▯  ▯     ▯    ÿÿ                @
$          ‡E▯dÅ$x7Å$x7Å$x79▯87
X4çwEšçw▯˝çwx▯æwo▯çwÛÉçwy6çv
%    ^     –     ]     =    p    +
ªc   £▯M▯▯ÿ0è9c   ¡▯M▯▯‰▯¡„O▯▯
 ÿ5]M▯▯ÿÖ▯MàQP‰E▯ÿÓ▯EÀPÿ5]M
…▯ÿÿÿPjTÿ5HJ▯▯ÿ▯¬▯  ▯…Àt▯j▯V
▯@▯▯▯u ;ôu▯…ÿu▯¡▯@▯▯j▯™_÷ÿJt
```

Binary File Access Modes



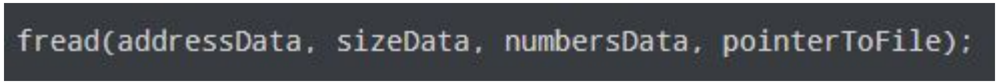
Mode	Description
rb	Open file in binary mode for reading only.
wb	Open file in binary mode for writing only. It creates the file if it does not exist. If the file exists, then it erases all the contents of the file.
ab	Open file in binary mode for appending data. Data is added to the end of the file. It creates the file if it does not exist.
rb+	Open file in binary mode for both reading and writing.
wb+	Open file in binary mode for both reading and writing. It creates the file if it does not exist. If the file exists, then it erases all the contents of the file.
ab+	Open a file in binary mode for reading and appending data. Data is added to the end of the file. It creates the file if it does not exist.

Add a "b" to the mode string, to open the file in **binary mode**.
Eg: r → rb

A **complete list** of allowed **file mode strings** can be found here

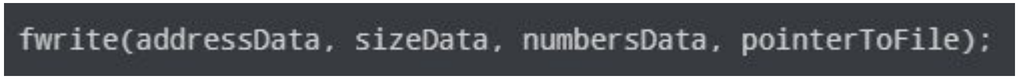
File Operations

- In C, the **typical operations** on files, either *text* or *binary*, include:
  - Creating a new file - **FILE ***
  - Opening an existing file- **fopen()**
  - Reading data from a file - **fread()**



```
fread(addressData, sizeData, numbersData, pointerToFile);
```

  - Writing data to a file - **fwrite()**



```
fwrite(addressData, sizeData, numbersData, pointerToFile);
```

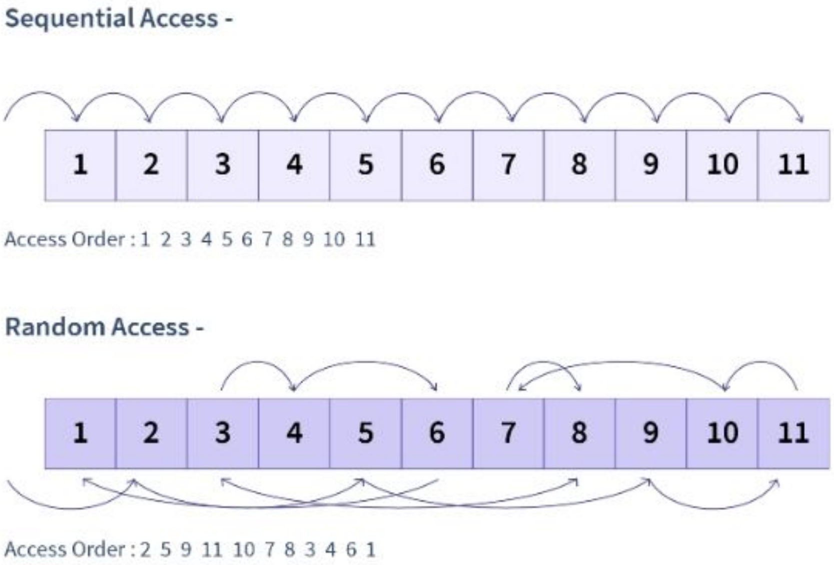
  - **Random Access - fseek(), ftell(), rewind()**
  - Closing a file - **fclose()**

DEMO for BINARY FILES

Random Access Functions in C

In C, there is no need to read data sequentially in binary files like we did for text files. This random access is supported using the following functions:

- **fseek():** It is used to move the reading control to different positions using fseek function.
- **ftell():** It tells the byte location of current position of cursor in file pointer.
- **rewind():** It moves the control to beginning of the file.



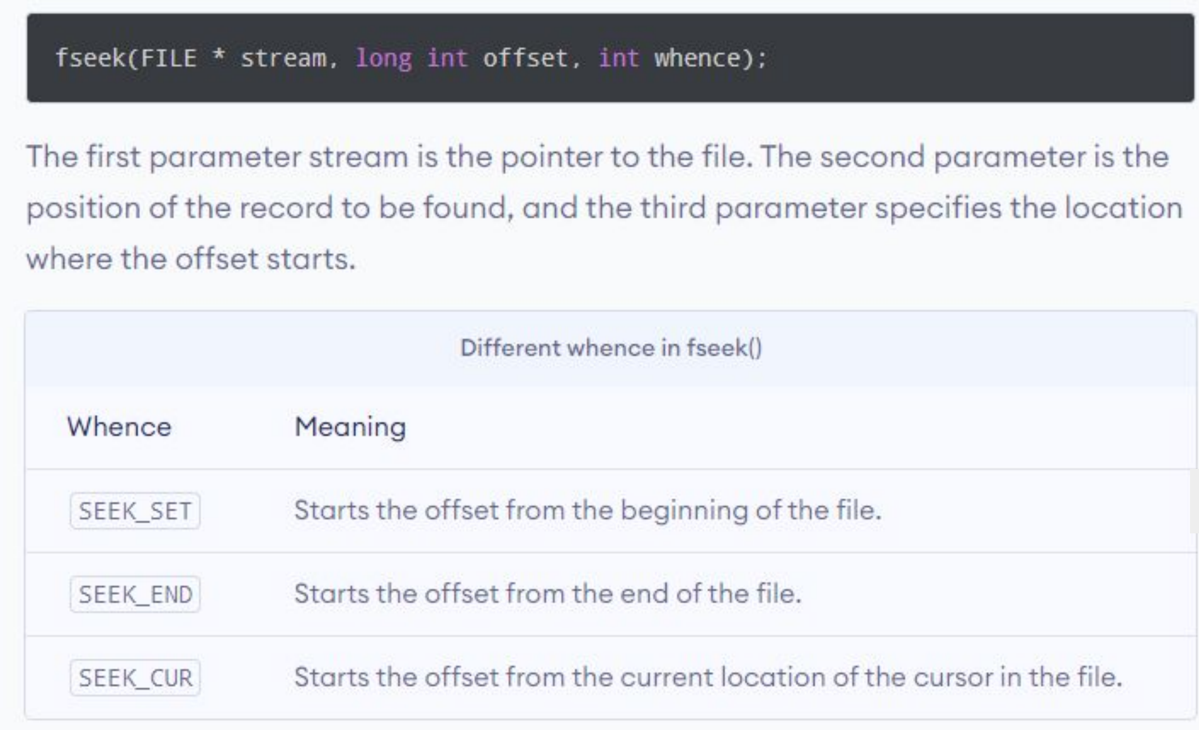
Sequential Access -

Access Order : 1 2 3 4 5 6 7 8 9 10 11

Random Access -

Access Order : 2 5 9 11 10 7 8 3 4 6 1

Tutorial

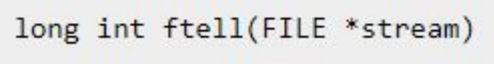


```
fseek(FILE * stream, long int offset, int whence);
```

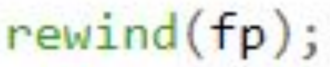
The first parameter stream is the pointer to the file. The second parameter is the position of the record to be found, and the third parameter specifies the location where the offset starts.

Different whence in fseek()

Whence	Meaning
SEEK_SET	Starts the offset from the beginning of the file.
SEEK_END	Starts the offset from the end of the file.
SEEK_CUR	Starts the offset from the current location of the cursor in the file.



```
long int ftell(FILE *stream)
```

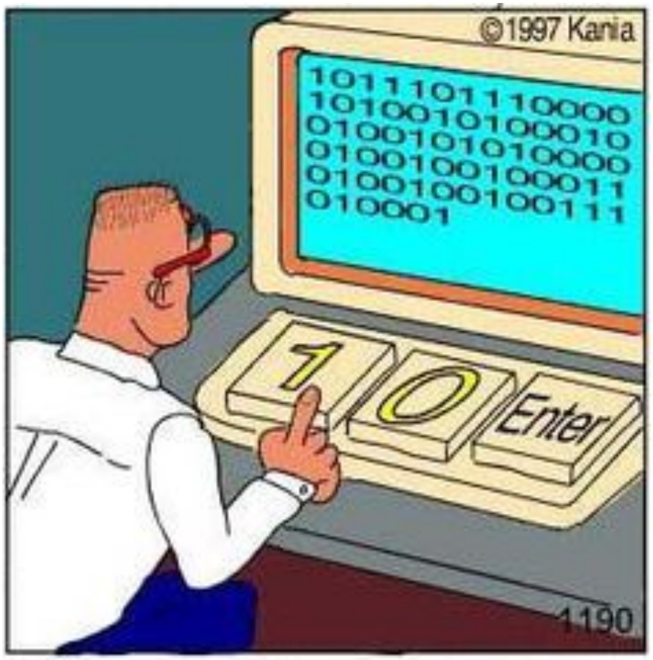


```
rewind(fp);
```

RANDOM ACCESS IN BINARY FILES

Next Session

FILES and STRUCTURES



Real programmers code in binary.

Self-referential Structures!!!!