COMP0204: Introduction to Programming for Robotics and AI

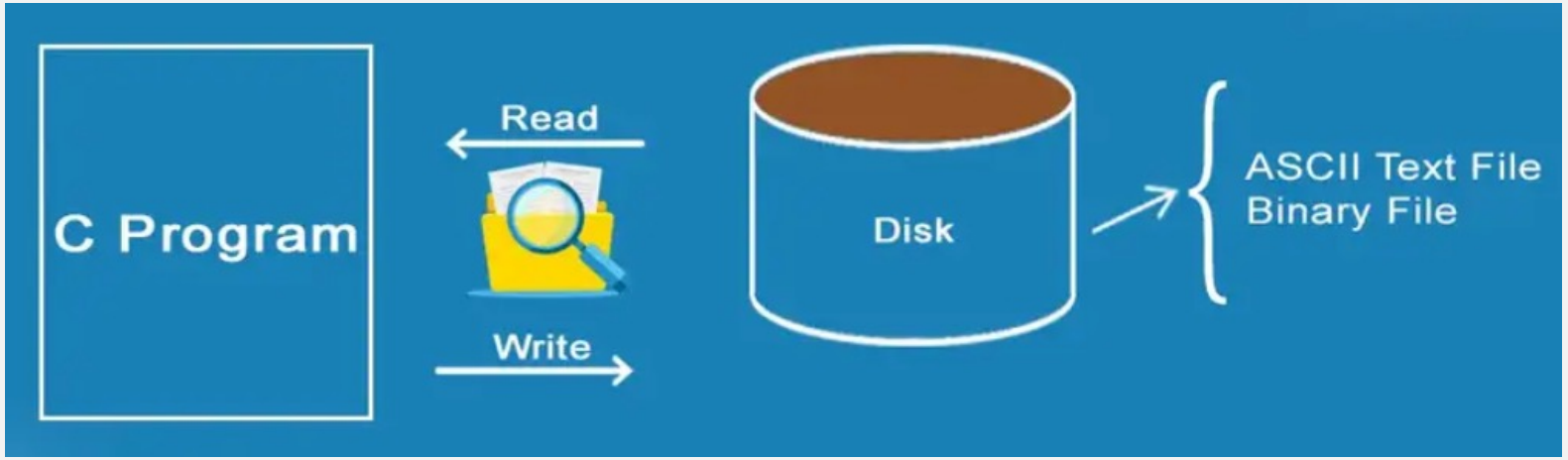
Lecture 8: File Handling in C

Course lead: Dr Sophia Bano

MEng Robotics and AI
UCL Computer Science

Today's lecture

- What are the **FILES** in C programming?
- Why we **need** FILES
- Different **types** of FILES
- **Sequential** vs **direct** access FILES
- How we can **create/write/read** or close any FILE
- Assessment 6



What are FILES in C?

- Storage of information in variables/arrays or pointers is (**temporary**).
- The data in these variables is lost when we shut down or switch off our system.

- Files are used for **permanent retention (store and retrieve)** of data on a computer's storage device, such as a hard disk, SSD, USB drive.
- **Examples of Filing:** Microsoft office, application data in smartphones, program logs, model weights, etc.

- File handling is implemented using the standard input/output functions provided in <stdio.h>

Type of FILES

Text type files

- Text type files are those type of **information/data**, which are easily **readable by humans**, Example include (*.txt, *.doc, *.ppt, *.cpp, *.c, *.h).
- We can read these types of FILES by opening in text editor.

Binary type files

- Binary type files can't be **readable or modifiable** by the humans. These contain non-textual data that is encoded in binary format.
- Only a **particular software** can open or view these types of FILES.
- Example include (*.gif, *.bmp, *.jpeg, *.exe, *.obj, *.dll).

Importance of File Handling in C Programming

- **Persistent Data Storage:** Files allow data to be stored outside of the program's memory, so it can be accessed even after the program has terminated.
- **Data Processing:** Many programs process large amounts of data, and files provide an efficient means of storing and processing this data.
- **Configuration Files:** Many programs use configuration files to store settings that are used each time the program is run.
- **Interprocess Communication:** Programs running on the same computer can use files to communicate with each other.

Accessing FILE

We can access the content of any FILE, by using two types

- **Sequential access**
- **Direct access (Random access)**

Sequential Access

- Files contents are accessed sequentially (**from first to desire content**).
- To access the location 101th, we must first traverse all contents from 0 to 100. Then the 101th location is access.
- (Normally slow when we want to access random contents in file). **Random access is especially access in Databases.**

Sequential Access FILE (Example)

- To access the data of the student of ID# 5.
- We have to follow the following 3 steps.
  1. **First open the file.**
  2. Must have to traverse all the contents of **ID# 1, 2, 3 and 4**. Then finally we can access **ID# 5** data.
  3. **Close the file.**

NIC#	Student Name	Marks
1	Emma Thompson	78
2	Jackson Patel	84
3	Mia Rodriguez	96
4	Olivia Kim	72
5	Noah Gupta	79
6	Chloe Anderson	90
7	Grace Harrison	75

Direct Access FILE

- **Direct Access**
  - In direct access, we can ***access any location data directly*** without ***traversing those contents*** which are ***stored before that location***.
  - Very ***FAST*** than normal sequential accessing.

Direct Accessing FILE (Example)

- Suppose, we want to access **NIC# 5** data.
- In direct access, we will move directly to the location **NIC# 5**.

NIC#	Student Name	Marks
1	Emma Thompson	78
2	Jackson Patel	84
3	Mia Rodriguez	96
4	Olivia Kim	72
5	Noah Gupta	79
6	Chloe Anderson	90
7	Grace Harrison	75

Steps in Processing a File

1. Create a FILE descriptor (FILE pointer point holds the disk location of the file we are working on).

```
FILE *fptr1;
```

1. Open the file, associating the stream name with the file name.
2. Read or write the data.
3. Close the file.

Opening a FILE

- Before we ***write or read*** a ***FILE from a disk***, ***we must open it***.
- Opening a ***FILE establishes*** a ***protocol (communication)*** between ***our program and the FILE system***.
- This ***protocol*** contains the ***following information:***
  - ***Which FILE*** we are going to ***access from which hard disk location***. Example ***"c:\\program files\\mydata\\Database.txt"***.
  - *What is the MODE of accessing, read or write.*

*This **communication system** is also called **FILE descriptor**.*

Opening a FILE

FileAccess.exe

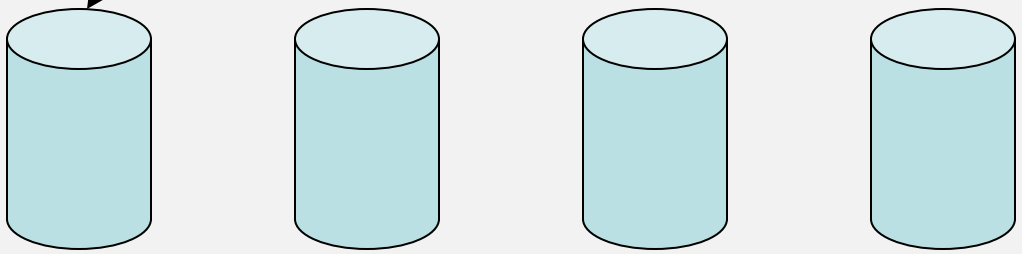
Want to read file C:\\firstFile.txt

Protocol (FILE descriptor)

The main question is:

1. How we can *access* this *FILE descriptor in C language*.

2. Every *FILE descriptor* is a pointer type information. *We can **store** this pointer in a **FILE type pointer***.



Hard disk C:\\

SSD D:\\

USB drive G:\\

Network drive H:\\

Files in C

- In C language, each ***FILE*** is simply a ***sequential stream of bytes***.

1	2	3	4	6	7	8	n-1	n
							...	End of file marker

- A ***FILE*** must ***first be opened properly*** before ***it accessed for reading or writing***.

- ***Successfully opening*** a ***FILE returns a pointer*** to (***i.e., the address of***) a **file descriptor**.

FILE pointer in C

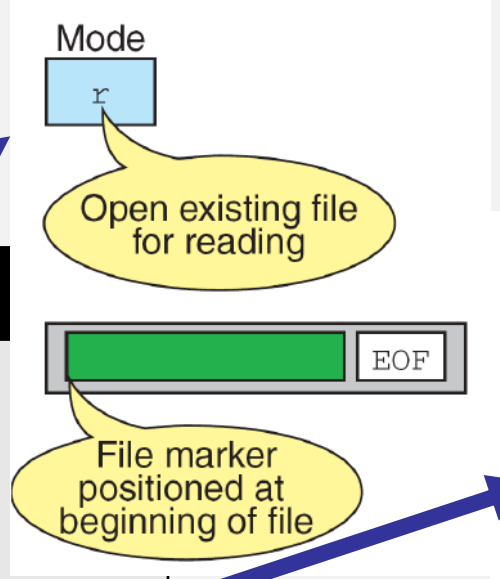
- ***The statement:***

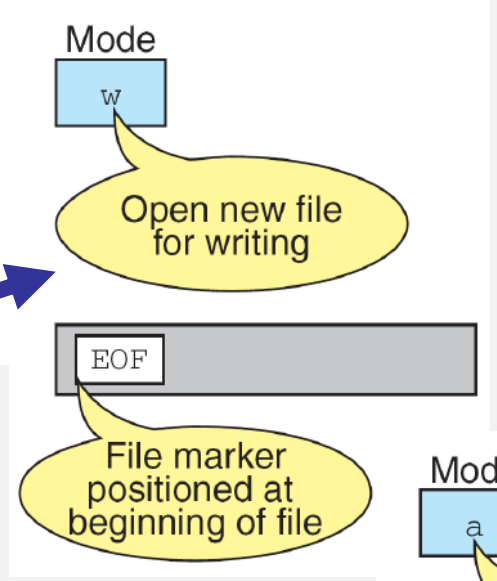
FILE *fptr1, *fptr2 ;

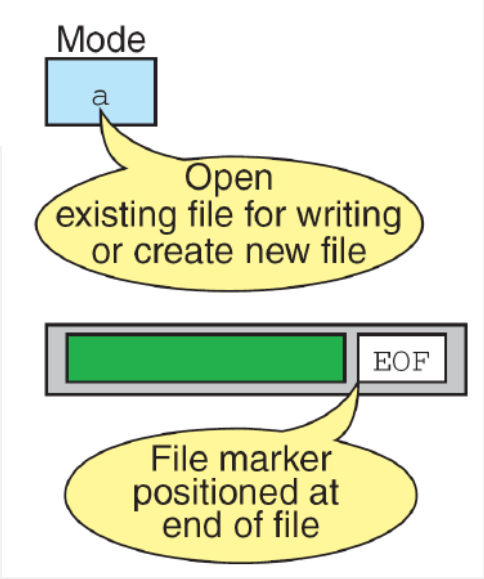
declares that ***fptr1*** and ***fptr2*** are pointer type variables of type **FILE**. They can contain the ***address of a file descriptors***.

File Open Modes

Mode	Meaning
r	Open text file in read mode • If file exists, the marker is positioned at beginning • If file doesn't exist, error returned **FILE *file = fopen("example.txt", "r");**
w	Open text file in write mode • If file exists, it is erased. • If file doesn't exist, it is created. **FILE *file = fopen("example.txt", "w");**
a	Open text file in append mode • If file exists, the marker is positioned at end. • If file doesn't exist, it is created. **FILE *file = fopen("example.txt", "a");**







Additional File Open Modes

Mode	Meaning
r+	Open for reading and writing, start at beginning • The file must already exist; otherwise, the function returns NULL.
w+	Open for reading and writing (overwrite file) • If the file already exists, its content is truncated. • If the file does not exist, a new file is created.
a+	Open for reading and writing (append if file exists) • New data being appended to the end of the file. • If the file does not exist, a new file is created.

Note: Use binary file modes to indicate that the file is to be treated as a binary file. Simply add a "b" to the end of the mode. E.g. “rb” – read in binary mode.

Opening FILE

- The statement:

```
FILE *fptr1;
            fptr1 = fopen( “d:\\mydata.txt", "r");
```

would open the file **d:\\mydata.txt** for ***input (reading)***.

- The statement:

```
FILE *fptr2;
            fptr2 = fopen(“d:\\results.txt", "w");
```

would open the file **d:\\results.txt** for ***output (writing)***.

Opening Text Files

- The statement:

```
FILE *fptr2;
fptr2 = fopen (“d:\\results.txt",“r+");
```

  would open the file **D:\\results.txt** for ***both reading and writing***.

- Once the ***FILE is open***, ***it stay open until you close it*** or ***end of the program reaches*** (which will close all files.)

Testing for Successful Open

- If the ***FILE*** was ***not able to be opened***, ***then the value returned*** by the ***fopen*** routine ***is NULL***.
- ***For example***, let's assume that the file **d:\\mydata.txt** ***does not exist***. Then:

```
FILE *fptr1;
fptr1 = fopen ( “d:\\mydata.txt", “w") ;
if (fptr1 == NULL){
  printf("File 'mydata' can’t be open”);
  return 1;
}
```

Basic file operations

- **fopen** - open a file- specify how its opened (read/write) and type (binary/text)
- **fclose** - close an opened file
- **fread** - read from a file
- **fwrite** - write to a file

For direct access

- **fseek** - move a file pointer to a desired point in a file.
- **ftell** - tell you where the file pointer is located (in terms of bytes from the start)
- **rewind** - sets the position to the beginning of the file.

Functions to read from/write to the file

Writing

- **fputc()** – writes a single character to a file
- **fputs()** – writes a string to a file
- **fprintf()** – writes formatted text to a file

Reading

- **fgetc()** – reads a single character from a file
- **fgets()** – reads a string from a file
- **fscanf()** – reads formatted input from a file

Writing in FILE

- ***We can write any contents (data) in the FILE*** using the following ***FILING function***:

fprintf (paramter1, paramter2, paramter3);

FILE pointer

Signature of the variable

Actual name of the variable

Writing in FILE - Example 1

```c
#include <stdio.h>
int main(void) {
    FILE *fp;
    int num = 93;

    fp = fopen("D:\\mydata.txt", "w");
    if (fp == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    fprintf(fp, "%d", num);
    fclose(fp);
    return 0;
}
```

The ***fprintf*** function will ***write value*** from the ***num (memory location)*** to ***hard disk location (“d:\\mydata.txt”).***

Closing FILES

- The statement:

  fclose (fptr) ;

  will close the file and release the file descriptor space from memory.

Writing in FILE - Example 2

```c
#include <stdio.h>
int main(void) {
    FILE *fp;
    int num = 93;
    float floatData = 34.63;

    fp = fopen("D:\\mydata.txt", "w");
    if (fp == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    fprintf(fp, "%d %f", num,floatData);

    fclose(fp);
    return 0;
}
```

The ***fprintf*** function will ***write value*** from the ***num & floatData (memory location)*** to ***hard disk location (“D:\\myfile.dat”).***

Writing character:

char myCharacter = ‘D’;
fprintf (fp, ”%d%f%c”, num,floatData,myCharacter);

Use of fputc:
fputc(myCharacter, fp);

Writing in FILE - Example 3

```c
#include <stdio.h>
int main(void) {
    FILE *fp;
    char str[20] = "Winter is coming";

    fp = fopen("D:\\mydata.txt", "w");
    if (fp == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    fprintf(fp, "%s", str);

    fclose(fp);
    return 0;
}
```

The ***fprintf*** function will ***write value*** from the string ***str (memory location)*** to ***hard disk location (“d:\\mydata.txt”).***

Use of fputs:

fputs(str, fp);

Writing in FILE (problem with “w” MODE)

```c
int main(void) {
    FILE *fp;
    int num = 353;

    fp = fopen("D:\\myfile.txt", "w"); // write mode
    if (fp == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    fprintf(fp, "%d", num);
    fclose(fp);
    return 0;
}
```

- ***What would be the final data*** in the ***FILE name myfile.txt*** when the ***above code executes 3 times***.
- The final data would be only ***(353)*** not ***(353353353)***.
- ***The reason is that***, with ***“w” MODE***, ***on every run the previous data is REMOVED***, ***and the new data is written from beginning of FILE***.

Solution Open in Append MODE “a+” or “w+”

```c
int main(void) {
    FILE *fp;
    int num = 353;

    fp = fopen("D:\\myfile.txt", "a+");
    if (fp == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    fprintf(fp, "%d", num);
    fclose(fp);
    return 0;
}
```

- *In append MODE, the previous data in the FILE is not removed on every new execution.*
- *The data in the FILE name “D:\\myfile.txt” on the 3 runs would be now (353353353).*

Reading from FILE

- ***We can read any contents (data) in the FILE*** using the following ***FILING function***:

FILE pointer

returnParameter fscanf (paramter1, paramter2, paramter3);

End of file indicator

Signature of the variable

Actual name of the variable

Reading from FILE - Example 4

```c
int main(void) {
    FILE *fp;
    int num2 = 0;

    fp = fopen( "D:\\mydata.txt" , "r+" ); // read and write
    if (fp == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    printf("num2 = %d\n", num2);

    fclose(fp);
    return 0;
}
```

The ***fscanf*** function will ***read value*** from the ***hard disk location (“D:\\mydata.txt”) to num2 (memory location)***.

Reading from FILE - Example 5 – use of rewind

```
int main(void) {
    FILE *fp;
    int num1 = 93, num2 = 0;

    fp = fopen( "D:\\mydata.txt" , "r+" ); // read and write
    if (fp == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    fprintf(fp, "%d ", num1);  // write to file
    rewind(fp);      // move file pointer to beginning of file
    fscanf(fp,"%d", &num2);    // read from file

    printf("num2 = %d\n", num2);

    fclose(fp);
    return 0;
}
```

Reading from FILE - Example 6 – use of fgets

```
int main(void) {
    FILE *fp;
    char str[20];

    fp = fopen( "D:\\mydata.txt" , "r+" ); // read and write
    if (fp == NULL) {
        printf("Error opening file\n");
        return 1;
    }

    fgets(str, 20, fp);  // use of fgets
    printf("%s", str);

    fclose(fp);
    return 0;
}
```

Reading byte by byte Information from FILE – Ex 7

write.c

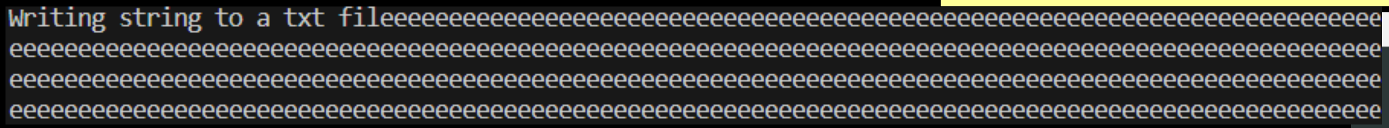
```
int main(void)
{
    FILE *fptr;
    fptr = fopen("D:\\myfile.txt", "w"); // write mode
    char string1[40];
    strcpy(string1, "Writing string to a txt file");
    fprintf(fptr, "%s", string1);
    fclose(fptr);
    return 0;
}
```

read.c

```
int main(void)
{
    FILE *fptr;
    fptr = fopen("D:\\myfile.txt", "r"); // read mode
    char character;
    while (1){
        fscanf(fptr, "%c", &character);
        printf("%c", character);
    }
    fclose(fptr);
    return 0;
}
```

The *problem with this code* is that it can *read out of the file data* from *your disk.*

Control it using 'End of file' marker.



```
Writing string to a txt fileeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
```

End of file

- The end-of-file (***feof***) indicator informs the program when there are no more data (no more bytes) to be processed.

```
if (feof(fp))
 {
     printf("End of file reached.\n");
 }
```

Reading byte by byte from FILE – Example 8

read.c

```c
#include <stdio.h>

void main()
{
    FILE * fp;
    char c;
    fp = fopen("D:\\myfile.txt", "r");
    if (fp == NULL)
        printf("File doesn't exist\n");
    else
    {   while(!feof(fp))
        {
            c = getc(fp); //get one character from the file
            putchar(c); //display it on the monitor
        }
     }
     fclose(fp);
}
```

Output:

```
Writing string to a txt file
```

Reading byte by byte from FILE – Example 9

read.c

```
void main()
{
    FILE * fp;
    char c;
    fp = fopen("D:\\myfile.txt", "r");
    if (fp == NULL)
        printf("File doesn't exist\n");
    else
    {
        do
        {
            c = getc(fp); //get one character from the file
            putchar(c); //display it on the monitor
        } (!feof(fp)) //repeat until feof (end of file)
     }
     fclose(fp);
}
```

Output:

```
Writing string to a txt file
```

Exercise

- Take the file ‘`myfile.txt`’ written by ‘*`write.c`*‘.
- Write a C program that append a sentence in ‘`myfile.txt`’.

Exercise

- Write a C program to create a new file ‘bookdetails.txt’ and write the following text to it.

Book: C Programming Absolute Beginner’s Guide

Author: Greg Perry

Book: Head First C: A Brain-Friendly Guide

Author: David Griffiths

Exercise – Read file, update data and write

- Write a C program to read the data from ‘bookdetails.txt’, delete all the vowels (a, e, i, o, u), and write the updated string to a new file ‘bookdetails_novowels.txt”.

Read/Write – Binary FILE

- Binary type files can't be **readable or modifiable** by the humans. These contain non-textual data that is encoded in binary format.
- Only a **particular software** can open or view these types of FILES.
- Example include (*.gif, *.bmp, *.jpeg, *.exe, *.obj, *.dll).

Function	Description
fwrite	Syntax: size_t fwrite(const void *ptr, size_t size, size_t count, FILE *stream); **Writes data** to a file in binary format. It takes a pointer to the data (ptr), the size of each element (size), the number of elements to write (count), and the file pointer (stream).
fread	size_t fread(void *ptr, size_t size, size_t count, FILE *stream); **Reads data** from a file in binary format. It takes a pointer to where the data will be stored (ptr), the size of each element (size), the number of elements to read (count), and the file pointer (stream).

Read/Write – Binary FILE - Example

```c
#include <stdio.h>

int main() {
    FILE *file;
    int data[] = {10, 20, 30, 40, 50};

    // Writing to a binary file
    file = fopen("D:\\binary_data.dat", "wb");
    fwrite(data, sizeof(int), sizeof(data) / sizeof(int), file);
    fclose(file);

    // Reading from the binary file
    file = fopen("D:\\binary_data.dat", "rb");
    int readData[5];
    fread(readData, sizeof(int), sizeof(readData) / sizeof(int), file);
    fclose(file);

    // Displaying the read data
    printf("Data read from the binary file:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", readData[i]);
    }

    return 0;
}
```

Direct Access FILE

- **Direct Access**
  - In direct access, we ***read or write data*** at ***any position*** within a file, not just sequentially from the beginning.
  - Very ***FAST*** than normal sequential accessing.

Direct Access FILE – Access functions

Function	Description
fwrite	**Syntax:** int fseek(FILE *stream, long offset, int whence); Moves the file position indicator to a specified location within the file. stream: A pointer to the FILE structure representing the file. offset: Number of bytes to move the indicator. whence: Specifies the reference position for the offset (e.g., SEEK_SET for the beginning of the file, SEEK_CUR for the current position, and SEEK_END for the end).
fread	**Syntax:** long ftell(FILE *stream); Returns the current file position indicator's position. stream: A pointer to the FILE structure representing the file.

Direct Access FILE - Usage

fseek

```
// Move to the beginning of the file
fseek(file, 0, SEEK_SET);

// Write data at the beginning of the file
int number1 = 42;
fwrite(&number1, sizeof(int), 1, file);
```

ftell

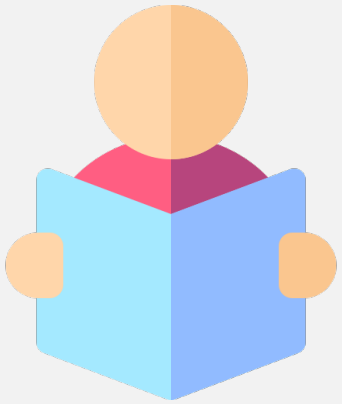
```
// Use ftell to get the current file position indicator
long position = ftell(file);
printf("\nCurrent file position after fseek: %ld\n", position);
```

Exercise – Direct Access FILE

- Write a C program that write alphabets A to Z into a file names letters.txt.
- It then loops backward through the file printing each of the letters from Z to A.

Solution:
In Chap 29 of C Programming for Absolute Beginner's Guide by Greg Perry and Dean Miller

Additional reading and coding



Book:

C Programming for Absolute Beginner's Guide by Greg Perry and Dean Miller

(E-book available in UCL Library)

Chapter 28: Saving Sequential Files to Your Computer

Chapter 29: Saving Random Files to Your Computer

Practice Exercises: https://www.w3resource.com/c-programming-exercises/file-handling/index.php