



Unit VIII

(File Handling)

Lecture Slide



AS2023



Objectives



By the end of this session, students will be able to:

- Explain file handling in C
- Use standard I/O file handling in C
- Understand the types of file operations
- Implement the Types of Modes.
- Create, Open, Read and Write on a Text file
- Create, Open, Read and Write on a Binary file
- Implement of fseek()



Introduction



What is file?

- A file is a container in computer storage devices used for storing data



Why files are needed?



- When a program is terminated, the entire data is lost.
- Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all.
- However, if you have a file containing all the data, you can easily access the contents of the file using a few commands in C.
- You can easily move your data from one computer to another without any changes.



Types of Files



- What types of files will be supported in C?
- When dealing with files, there are two types of files you should know about:
 1. *Text files*
 2. *Binary files*



1. Text Files



- Text files are the normal **.txt** files. You can easily create text files using any simple text editors such as *Notepad*.
- When you open those files, you'll see all the contents within the file as plain text.
- You can easily edit or delete the contents.
- They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.



2. Binary Files



- Binary files are mostly the **.bin** files in your computer.
- Instead of storing data in plain text, they store it in the binary form (**0's** and **1's**).
- They can hold a higher amount of data, are not readable easily, and provides better security than text files.



File Operations



In C, you can perform four major operations on files, either text or binary:

1. *Creating a new file*
2. *Opening an existing file*
3. *Reading from and writing information to a file*
4. *Closing a file*



Working with Files



- When working with files, you need to declare a pointer of type file.
- This declaration is needed for communication between the file and the program.

```
FILE *fptr;
```



Opening a File - for Creation and Edit



- Opening a file is performed using the **fopen()** function defined in the **stdio.h** header file.
- The syntax for opening a file in standard I/O is:

```
ptr = fopen("fileopen", "mode");
```

- For example:

```
fopen("E:\\cprogram\\newprogram.txt", "w");
```

```
fopen("E:\\cprogram\\oldprogram.bin", "rb");
```



Opening a File - for Creation and Edit (contd...)



1. Let's suppose the file **newprogram.txt** doesn't exist in the location **E:\cprogram**.
 - The first function creates a new file named **newprogram.txt** and opens it for writing as per the mode **'w'**.
 - The writing mode allows you to create and edit (overwrite) the contents of the file.
2. Now let's suppose the second binary file **oldprogram.bin** exists in the location **E:\cprogram**.
 1. The second function opens the existing file for reading in binary mode **'rb'**.
 2. The reading mode only allows you to read the file, you cannot write into the file.



Opening Modes in Standard I/O



Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.



Opening Modes in Standard I/O (contd....)



Mode	Meaning of Mode	During Inexistence of file
a	Open for append. Data is added to the end of the file.	If the file does not exist, it will be created.
ab	Open for append in binary mode. Data is added to the end of the file.	If the file does not exist, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.



Opening Modes in Standard I/O (contd....)



Mode	Meaning of Mode	During Inexistence of file
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exist, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.



Closing a File



- The file (both text and binary) should be closed after reading/writing.
- Closing a file is performed using the `fclose()` function.

`fclose(fptr) ;`

- Here, `fptr` is a file pointer associated with the file to be closed.



Reading and writing to a Text file



- For reading and writing to a text file, we use the functions ***fprintf()*** and ***fscanf()***.
- They are just the file versions of ***printf()*** and ***scanf()***.
- The only difference is that ***fprint()*** and ***fscanf()*** expects a pointer to the structure **FILE**.



Write to a Text File

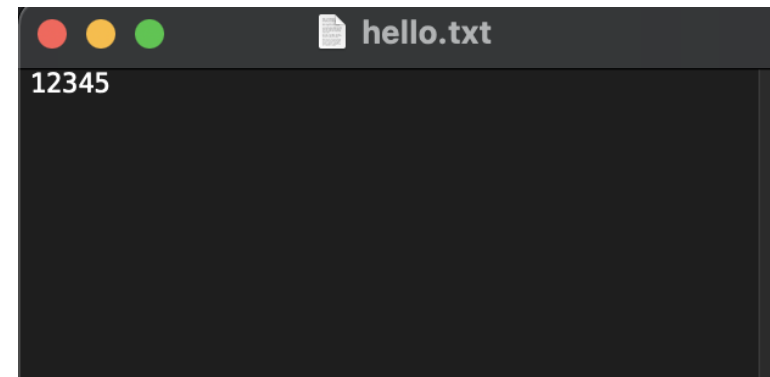


```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int num;
    FILE *fptr;
    fptr = fopen(" C:\\\\ hello.txt", "w");
    if(fptr == NULL) {
        printf("Error!");
        exit(1);
    }
    printf("Enter num: ");
    scanf("%d", &num);

    fprintf(fptr, "%d", num);
    fclose(fptr);
    return 0;
}
```

Output Sample:

Enter num: 12345





Read from a Text File

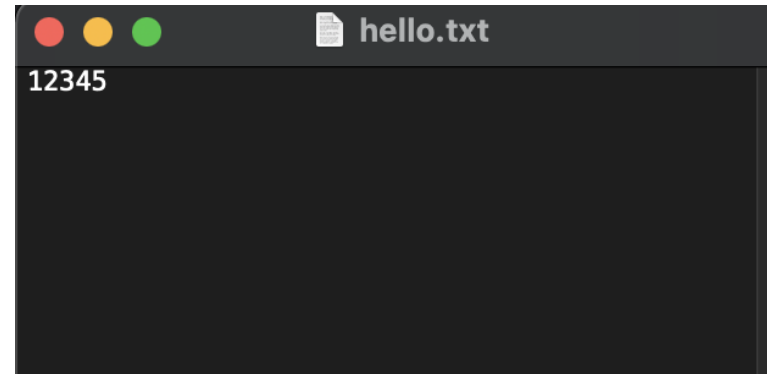


```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int num;
    FILE *fptr;

    if ((fptr = fopen("C:\\\\hello.txt", "r")) == NULL) {
        printf("Error! opening file");
        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    fscanf(fptr, "%d", &num);
    printf("Value of n = %d", num);
    fclose(fptr);

    return 0;
}
```



Output Sample:
Value of n = 12345



Reading and writing to a Binary file



- Functions ***fread()*** and ***fwrite()*** are used for reading from and writing to a file on the disk respectively in case of **binary** files.



Getting data using *fseek()*



- Why you need ***fseek()***?
 - If you have many records inside a file and need to access a record at a specific position, you need to loop through all the records before it to get the record.
 - This will **waste** a lot of memory and operation time.
- An easier way to get to the required data can be achieved using ***fseek()***.
- As the name suggests, ***fseek()*** seeks the cursor to the given record in the file.



Syntax of *fseek()*



`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 <code>fseek()</code>	
Whence Constant	Meaning
<code>SEEK_SET</code>	Starts the offset from the beginning of the file.
<code>SEEK_END</code>	Starts the offset from the end of the file.
<code>SEEK_CUR</code>	Starts the offset from the current location of the cursor in the file.



Example 1: fseek()



```
#include <stdio.h>

int main () {
    FILE *fp;
    fp = fopen("file.txt","w+");
    fputs("This is tutorialspoint.com", fp);
    // fprintf(fp, "This is tutorialspoint.com");
    fseek( fp, 7, SEEK_SET );
    fputs(" C Programming Language", fp);
    // fprintf(fp, " C Programming Language");
    fclose(fp);
    return(0);
}
```

Note: Initially program creates the file **file.txt** and writes **"This is tutorialspoint.com"** but later we had reset the write pointer at 7th position from the beginning and used *puts()* statement which overwrite the file with **"This is C Programming Language"**



Thank you