# CL1002 Programming Fundamentals

# LAB 12

## Filing in C

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

File handing in C is the process in which we create, open, read, write, and close operations on a file. C language provides different functions such as fopen(), fwrite(), fread(), fseek(), fprintf(), etc. to perform input, output, and many different C file operations in our program.

# Why do we need File Handling in C?

So far, the operations using the C program are done on a prompt/terminal which is not stored anywhere. The output is deleted when the program is closed. But in the software industry, most programs are written to store the information fetched from the program. The use of file handling is exactly what the situation calls for.

To understand why file handling is important, let us look at a few features of using files:

- **Reusability:** The data stored in the file can be accessed, updated, and deleted anywhere and anytime providing high reusability.
- **Portability:** Without losing any data, files can be transferred to another in the computer system. The risk of flawed coding is minimized with this feature.
- **Efficient:** A large amount of input may be required for some programs. File handling allows you to easily access a part of a file using few instructions which saves a lot of time and reduces the chance of errors.
- **Storage Capacity:** Files allow you to store a large amount of data without having to worry about storing everything simultaneously in a program.

## Types of Files in C

A file can be classified into two types based on the way the file stores the data. They are as follows:

- **Text Files**
- **Binary Files**

### 1. Text Files

A text file contains data in the **form of ASCII characters** and is generally used to store a stream of characters.

- Each line in a text file ends with a new line character ('\n').
- It can be read or written by any text editor.
- They are generally stored with **.txt** file extension.
- Text files can also be used to store the source code.

## 2. Binary Files

A binary file contains data in **binary form (i.e. 0's and 1's)** instead of ASCII characters. They contain data that is stored in a similar manner to how it is stored in the main memory.

- Binary files can be created only from within a program and their contents can only be read by a program.
- More secure as they are not easily readable.
- They are generally stored with **.bin** file extension.

## C File Operations

C file operations refer to the different possible operations that we can perform on a file in C such as:

1. Creating a new file – **fopen() with attributes as "a" or "a+" or "w" or "w+"**
2. Opening an existing file – **fopen()**
3. Reading from file – **fscanf() or fgets()**
4. Writing to a file – **fprintf() or fputs()**
5. Moving to a specific location in a file – **fseek(), rewind()**
6. Closing a file – **fclose()**

The highlighted text mentions the C function used to perform the file operations.

## Functions for C File Operations

| File operation | Declaration & Description |
|---|---|
| **fopen() - To open a file** | Declaration: FILE *fopen (const char *filename, const char *mode)<br>fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist.<br>FILE *fp;<br>fp=fopen ("filename", "'mode");<br>Where,<br>fp – file pointer to the data type "FILE".<br>filename – the actual file name with full path of the file.<br>mode – refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+. Please refer below the description for these mode of operations. |
| **fclose() - To close a file** | Declaration: int fclose(FILE *fp);<br>fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below.<br>fclose (fp); |
| **fgets() - To read a file** | Declaration: char *fgets(char *string, int n, FILE *fp)<br>fgets function is used to read a file line by line. In a C program, we use fgets function as below.<br>fgets (buffer, size, fp);<br>where,<br>buffer – buffer to put the data in.<br>size – size of the buffer<br>fp – file pointer |
| **fprintf() - To write into a file** | Declaration:<br>int fprintf(FILE *fp, const char *format, ...);fprintf() function writes string into a file pointed by fp. In a C program, we write string into a file as below. fprintf (fp, "some data"); or<br>fprintf (fp, "text %d", variable_name); |

## File Pointer in C

A file pointer is a reference to a particular position in the opened file. It is used in file handling to perform all file operations such as read, write, close, etc. We use the **FILE** macro to declare the file pointer variable. The FILE macro is defined inside **<stdio.h>** header file.

Syntax of File Pointer

```
FILE* pointer_name;
```

File Pointer is used in almost all the file operations in C.

## Open a File in C

For opening a file in C, the fopen() function is used with the filename or file path along with the required access modes.

Syntax of fopen()

```
FILE* fopen(const char *file_name, const char *access_mode);
```

Parameters

- *file_name:* name of the file when present in the same directory as the source file. Otherwise, full path.
- *access_mode:* Specifies for what operation the file is being opened.

Return Value

- If the file is opened successfully, returns a file pointer to it.
- If the file is not opened, then returns NULL.

File opening modes in C

File opening modes or access modes specify the allowed operations on the file to be opened. They are passed as an argument to the fopen() function. Some of the commonly used file access modes are listed below:

| Opening Modes | Description |
|---|---|
| **r** | Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the first character in it. If the file cannot be opened fopen( ) returns NULL. |
| **rb** | Open for reading in binary mode. If the file does not exist, fopen( ) returns NULL. |
| **w** | Open for writing in text mode. If the file exists, its contents are overwritten. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file. |
| **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. |
| **a** | Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the last character in it. It opens only in the append mode. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file. |
| **ab** | Open to append in binary mode. Data is added to the end of the file. If the file does not exist, it will be created. |
| **r+** | Searches file. It is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the first character in it. Returns NULL, if unable to open the file. |

| Opening Modes | Description |
|---|---|
| **rb+** | Open for both reading and writing in binary mode. If the file does not exist, fopen( ) returns NULL. |
| **w+** | Searches file. If the file exists, its contents are overwritten. If the file doesn't exist a new file is created. Returns NULL, if unable to open the file. |
| **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+** | Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the last character in it. It opens the file in both reading and append mode. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file. |
| **ab+** | Open for both reading and appending in binary mode. If the file does not exist, it will be created. |

As given above, if you want to perform operations on a binary file, then you have to append 'b' at the last. For example, instead of "w", you have to use "wb", instead of "a+" you have to use "a+b".

Example of Opening a File

```c
// C Program to illustrate file opening
#include <stdio.h>
#include <stdlib.h>

int main()
{
    // file pointer variable to store the value returned by
    // fopen
    FILE* fptr;

    // opening the file in read mode
    fptr = fopen("filename.txt", "r");

    // checking if the file is opened successfully
    if (fptr == NULL) {
        printf("The file is not opened. The program will "
               "now exit.");
        exit(0);
    }

    return 0;
}
```

**Output**

```
The file is not opened. The program will now exit.
```

The file is not opened because it does not exist in the source directory. But the fopen() function is also capable of creating a file if it does not exist. It is shown below.

## Create a File in C

The fopen() function can not only open a file but also can create a file if it does not exist already. For that, we have to use the modes that allow the creation of a file if not found such as w, w+, wb, wb+, a, a+, ab, and ab+.

```
FILE *fptr;
fptr = fopen("filename.txt", "w");
Example of Opening a File
// C Program to create a file
#include <stdio.h>
#include <stdlib.h>

int main()
{
    // file pointer
    FILE* fptr;

    // creating file using fopen() access mode "w"
    fptr = fopen("file.txt", "w");

    // checking if the file is created
    if (fptr == NULL) {
        printf("The file is not opened. The program will "
               "exit now");
        exit(0);
    }
    else {
        printf("The file is created Successfully.");
    }

    return 0;
}
```

**Output**

```
The file is created Successfully.
```

## Reading From a File

The file read operation in C can be performed using functions fscanf() or fgets(). Both the functions performed the same operations as that of scanf and gets but with an additional parameter, the file

pointer. There are also other functions we can use to read from a file. Such functions are listed below:

| Function | Description |
|---|---|
| **fscanf()** | Use formatted string and variable arguments list to take input from a file. |
| **fgets()** | Input the whole line from the file. |
| **fgetc()** | Reads a single character from the file. |
| **fgetw()** | Reads a number from a file. |
| **fread()** | Reads the specified bytes of data from a binary file. |

So, it depends on you if you want to read the file line by line or character by character.

Example:

```
FILE * fptr;
fptr = fopen(“fileName.txt”, “r”);
fscanf(fptr, "%s %s %s %d", str1, str2, str3, &year);
char c = fgetc(fptr);
```

The getc() and some other file reading functions return EOF (End Of File) when they reach the end of the file while reading. EOF indicates the end of the file and its value is implementation-defined.

**Note:** One thing to note here is that after reading a particular part of the file, the file pointer will be automatically moved to the end of the last read character.

## Write to a File

The file write operations can be performed by the functions fprintf() and fputs() with similarities to read operations. C programming also provides some other functions that can be used to write data to a file such as:

| Function | Description |
|---|---|
| **fprintf()** | Similar to printf(), this function use formatted string and varible arguments list to print output to the file. |
| **fputs()** | Prints the whole line in the file and a newline at the end. |
| **fputc()** | Prints a single character into the file. |
| **fputw()** | Prints a number to the file. |

| Function | Description |
|---|---|
| **fwrite()** | This function write the specified amount of bytes to the binary file. |

Example:

```
FILE *fptr ;
fptr = fopen(“fileName.txt”, “w”);
fprintf(fptr, "%s %s %s %d", "We", "are", "in", 2012);
fputc("a", fptr);
```

## Closing a File

The fclose() function is used to close the file. After successful file operations, you must always close a file to remove it from memory.

```
Syntax of fclose()
fclose(file_pointer);
```

where the *file_pointer* is the pointer to the opened file.

Example:

```
FILE *fptr ;
fptr= fopen(“fileName.txt”, “w”);
---------- Some file Operations -------
fclose(fptr);
```

## Examples of File Handing in C

Example 1: Program to Create a File, Write in it, And Close the File

```
// C program to Open a File,
// Write in it, And Close the File
#include <stdio.h>
#include <string.h>

int main()
{

    // Declare the file pointer
    FILE* filePointer;

    // Get the data to be written in file
    char dataToBeWritten[50] = "Welcome "
                               "PF Batch 2k23";

    // Open the existing file PFfile.c using fopen()
    // in write mode using "w" attribute
```

```
    filePointer = fopen("PFfile.c", "w");

    // Check if this filePointer is null
    // which maybe if the file does not exist
    if (filePointer == NULL) {
        printf("PFfile.c file failed to open.");
    }
    else {

        printf("The file is now opened.\n");

        // Write the dataToBeWritten into the file
        if (strlen(dataToBeWritten) > 0) {

            // writing in the file using fputs()
            fputs(dataToBeWritten, filePointer);
            fputs("\n", filePointer);
        }

        // Closing the file using fclose()
        fclose(filePointer);

        printf("Data successfully written in file "
            "PFfile.c\n");
        printf("The file is now closed.");
    }

    return 0;
}
```

**Output**

```
The file is now opened.
Data successfully written in file PFfile.c
The file is now closed.
```

This program will create a file named PFfile.c in the same directory as the source file which will contain the following text: "**Welcome PF Batch 2k23**".

Example 2: Program to Open a File, Read from it, And Close the File

```
// C program to Open a File,
// Read from it, And Close the File
#include <stdio.h>
#include <string.h>

int main()
{
```

```c
    // Declare the file pointer
    FILE* filePointer;

    // Declare the variable for the data to be read from
    // file
    char dataToBeRead[50];

    // Open the existing file PFfile.c using fopen()
    // in read mode using "r" attribute
    filePointer = fopen("PFfile.c", "r");

    // Check if this filePointer is null
    // which maybe if the file does not exist
    if (filePointer == NULL) {
        printf("PFfile.c file failed to open.");
    }
    else {

        printf("The file is now opened.\n");

        // Read the dataToBeRead from the file
        // using fgets() method
        while (fgets(dataToBeRead, 50, filePointer)
               != NULL) {

            // Print the dataToBeRead
            printf("%s", dataToBeRead);
        }

        // Closing the file using fclose()
        fclose(filePointer);

        printf(
            "Data successfully read from file PFfile.c\n");
        printf("The file is now closed.");
    }
    return 0;
}
```

**Output**

```
The file is now opened.
Welcome PF batch 2k23
Data successfully read from file PFfile.c
The file is now closed.
```

This program reads the text from the file named PFfile.c which we created in the previous example and prints it in the console.

## Read and Write in a Binary File

Till now, we have only discussed text file operations. The operations on a binary file are similar to text file operations with little difference.

Opening a Binary File

To open a file in binary mode, we use the rb, rb+, ab, ab+, wb, and wb+ access mode in the fopen() function. We also use the .bin file extension in the binary filename.

**Example**

```
fptr = fopen("filename.bin", "rb");
```

Write to a Binary File

We use fwrite() function to write data to a binary file. The data is written to the binary file in the from of bits (0's and 1's).

**Syntax of fwrite()**

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *file_pointer);
```

**Parameters:**

- **ptr:** pointer to the block of memory to be written.
- **size:** size of each element to be written (in bytes).
- **nmemb:** number of elements.
- **file_pointer**: FILE pointer to the output file stream.

**Return Value:**

- Number of objects written.

**Example:** Program to write to a Binary file using fwrite()

```
// C program to write to a Binary file using fwrite()
#include <stdio.h>
#include <stdlib.h>
struct threeNum {
    int n1, n2, n3;
};
int main()
{
    int n;
    // Structure variable declared here.
    struct threeNum num;
```

```
    FILE* fptr;
    if ((fptr = fopen("C:\\program.bin", "wb")) == NULL) {
        printf("Error! opening file");
        // If file pointer will return NULL
        // Program will exit.
        exit(1);
    }
    int flag = 0;
    // else it will return a pointer to the file.
    for (n = 1; n < 5; ++n) {
        num.n1 = n;
        num.n2 = 5 * n;
        num.n3 = 5 * n + 1;
        flag = fwrite(&num, sizeof(struct threeNum), 1,
                      fptr);
    }

    // checking if the data is written
    if (!flag) {
        printf("Write Operation Failure");
    }
    else {
        printf("Write Operation Successful");
    }

    fclose(fptr);

    return 0;
}
```

**Output**

```
Write Operation Successful
```

Reading from Binary File

The fread() function can be used to read data from a binary file in C. The data is read from the file in the same form as it is stored i.e. binary form.

### Syntax of fread()

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *file_pointer);
```

**Parameters:**

- **ptr:** pointer to the block of memory to read.
- **size:** the size of each element to read(in bytes).

- **nmemb:** number of elements.
- **file_pointer**: FILE pointer to the input file stream.

**Return Value:**

- Number of objects written.

**Example:** Program to Read from a binary file using fread()

```c
// C Program to Read from a binary file using fread()
#include <stdio.h>
#include <stdlib.h>
struct threeNum {
    int n1, n2, n3;
};
int main()
{
    int n;
    struct threeNum num;
    FILE* fptr;
    if ((fptr = fopen("C:\\program.bin", "rb")) == NULL) {
        printf("Error! opening file");
        // If file pointer will return NULL
        // Program will exit.
        exit(1);
    }
    // else it will return a pointer to the file.
    for (n = 1; n < 5; ++n) {
        fread(&num, sizeof(struct threeNum), 1, fptr);
        printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2,
               num.n3);
    }
    fclose(fptr);

    return 0;
}
```

**Output**

```
n1: 1   n2: 5   n3: 6
n1: 2   n2: 10  n3: 11
n1: 3   n2: 15  n3: 16
n1: 4   n2: 20  n3: 21
```

## fseek() in C

If we have multiple records inside a file and need to access a particular record that is at a specific position, so we need to loop through all the records before it to get the record. Doing this will waste a lot of memory and operational time. To reduce memory consumption and operational time we can use fseek() which provides an easier way to get to the required data. fseek() function in C seeks the cursor to the given record in the file.

Syntax for fseek()

```
int fseek(FILE *ptr, long int offset, int pos);
```

Example of fseek()

```
// C Program  to demonstrate the use of fseek() in C
#include <stdio.h>

int main()
{
    FILE* fp;
    fp = fopen("test.txt", "r");

    // Moving pointer to end
    fseek(fp, 0, SEEK_END);

    // Printing position of pointer
    printf("%ld", ftell(fp));

    return 0;
}
```

**Output**

```
81
```

## rewind() in C

The rewind() function is used to bring the file pointer to the beginning of the file. It can be used in place of fseek() when you want the file pointer at the start.

```
Syntax of rewind()
rewind (file_pointer);
```

Example

```
// C program to illustrate the use of rewind
#include <stdio.h>

int main()
{
    FILE* fptr;
```

```c
    fptr = fopen("file.txt", "w+");
    fprintf(fptr, "Batch 2023\n");

    // using rewind()
    rewind(fptr);

    // reading from file
    char buf[50];
    fscanf(fptr, "%[^\n]s", buf);

    printf("%s", buf);

    return 0;
}
```

**Output**

```
Batch 2023
```

## More Functions for C File Operations

The following table lists some more functions that can be used to perform file operations or assist in performing them.

| Functions | Description |
|---|---|
| fopen() | It is used to create a file or to open a file. |
| fclose() | It is used to close a file. |
| fgets() | It is used to read a file. |
| fprintf() | It is used to write blocks of data into a file. |
| fscanf() | It is used to read blocks of data from a file. |
| getc() | It is used to read a single character to a file. |
| putc() | It is used to write a single character to a file. |
| fseek() | It is used to set the position of a file pointer to a mentioned location. |
| ftell() | It is used to return the current position of a file pointer. |
| rewind() | It is used to set the file pointer to the beginning of a file. |

| Functions | Description |
|---|---|
| putw() | It is used to write an integer to a file. |
| getw() | It is used to read an integer from a file. |

## getc() Function in C

The C getc() function is used to read a single character from the given file stream. It is implemented as a macro in **<stdio.h>** header file.

```
Syntax of getc()
int getc(FILE* stream);
```

### Parameters

- **stream**: It is a pointer to a file stream to read the data from.

### Return Value

- It returns the character read from the file stream.
- If some error occurs or the End-Of-File is reached, it returns EOF.

## feof() Function in C

The feof() function is used to check whether the file pointer to a stream is pointing to the end of the file or not. It returns a non-zero value if the end is reached, otherwise, it returns 0.

```
Syntax of feof()
int feof(FILE* stream);
```

### Parameters

- It returns the character read from the file stream.

### Return Value

- If the end-of-file indicator has been set for the stream, the function returns a non-zero value (usually 1). Otherwise, it returns 0.

Example of feof()

In the program, returned value of getc() is compared with EOF first, then there is another check using feof(). By putting this check, we make sure that the program prints *"End of file reached"* only

if the end of the file is reached. And if getc() returns EOF due to any other reason, then the program prints *"Something went wrong"*.

```c
// C program to demonstrate the use of getc(), feof() and
// EOF
#include <stdio.h>

int main()
{
    // Open the file "test.txt" in read mode
    FILE* fp = fopen("test.txt", "r");
    // Read the first character from the file
    int ch = getc(fp);

    // Loop until the end of the file is
    // reached
    while (ch != EOF) {
        /* display contents of file on screen */
        putchar(ch);

        // Read the next character from the file
        ch = getc(fp);
    }

    // Check if the end-of-file indicator is
    // set for the file
    if (feof(fp))
        printf("\n End of file reached.");
    else
        printf("\n Something went wrong.");

    // Close the file
    fclose(fp);

    // Wait for a keypress
    getchar();
    return 0;
}
```

**If the file "test.txt" contains the following data:**

```
PF Batch 2023
```

**Output**

```
PF Batch 2023
End of file reached.
```

## fgets()

The fgets() 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.

Syntax

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

Parameters

- **str:** Pointer to an array of chars where the string read is copied.
- **n:** Maximum number of characters to be copied into str (including the terminating null character).
- ***stream:** Pointer to a FILE object that identifies an input stream.

**Note: stdin** can be used as argument to read from the standard input.

Return Value

- The fgets() function returns a pointer to the string where the input is stored.

Features of fgets()

- It follows some parameters such as Maximum length, buffer, and input device reference.
- It is **safe** to use because it checks the array bound.
- It keeps on reading until a new line character is encountered or the maximum limit of the character array.

Example of fgets()

Let's say the maximum number of characters is 15 and the input length is greater than 15 but still fgets() will read only 15 characters and print it.

```c
// C program to illustrate fgets()
#include <stdio.h>
#define MAX 15

int main()
{
    // defining buffer
    char buf[MAX];

    // using fgets to take input from stdin
    fgets(buf, MAX, stdin);
    printf("string is: %s\n", buf);

    return 0;
}
```

Since fgets() reads input from the user, we need to provide input during runtime.

```
Input:
Hello and welcome to Programming Fundamentals
Output:
string is: Hello and welc
```

## gets()

Reads characters from the standard input (stdin) and stores them as a C string into str until a newline character or the end-of-file is reached.

- It is not safe to use because it does not check the array bound.
- It is used to read strings from the user until a newline character is not encountered.

Syntax

```
char *gets( char *str );
```

Parameters

- **str:** Pointer to a block of memory (array of char) where the string read is copied as a C string.

Return Value

- The function returns a pointer to the string where input is stored.

Example of gets()

Suppose we have a character array of 15 characters and the input is greater than 15 characters, gets() will read all these characters and store them into a variable. Since, gets() does not check the maximum limit of input characters, at any time compiler may return buffer overflow error.

```
// C program to illustrate
// gets()
#include <stdio.h>
#define MAX 15

int main()
{
    // defining buffer
    char buf[MAX];

    printf("Enter a string: ");

    // using gets to take string from stdin
    gets(buf);
    printf("string is: %s\n", buf);

    return 0;
}
```

```
```

Since gets() reads input from the user, we need to provide input during runtime.

**Input:**
```
Hello and welcome to Programming Fundamentals
```

**Output:**
```
Hello and welcome to Programming Fundamentals
```

## Conclusion

Both fgets() and gets() functions can be used for reading string input from standard input. The main difference between fgets() function and gets() function is that fgets() function allows the user to specify the maximum number of characters to read and we can also change the input stream to any file in fgets().

# What is return type of getchar(), fgetc() and getc() ?

In C, return type of getchar(), fgetc() and getc() is int (not char). So it is recommended to assign the returned values of these functions to an integer type variable.

```c
char ch;  /* May cause problems */
while ((ch = getchar()) != EOF)
{
   putchar(ch);
}
```

Here is a version that uses integer to compare the value of getchar().

```c
int in;
while ((in = getchar()) != EOF)
{
   putchar(in);
}
```