**VISION INSTITUTE OF MANAGEMENT**

**C PROGRAMMING**

**BCA 1st YEAR/2nd SEM**

**UNIT-6(FILE HANDLING)**

# **Files in C Programming**

## **Introduction**

* A collection of data which is stored on a secondary device like a hard disk is known as **a file**.
* A file is generally used as real-life applications that contain a large amount of data.

**There are two problems with such applications:**

1. It is time-consuming and unmanageable for handling such huge amount of data,
2. When the I/O terminal is used the entire data is lost if the program is terminated or the computer is being turned off. So it is a compulsion for storing the data on a permanent device.

**Files are divided into two types-**

1. **Stream-oriented –** they are standard or high-level files. They are easier to work with than the system-oriented data-files and are used more commonly.
2. **System-oriented –** they are low-level files.

**The library functions which are used for operating the files are:**

1. **High-level file I/O functions –** they do their own buffer management.
2. **Low-level file I/O functions –** the buffer management is done by the programmer.

## **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**

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. Closing a file
4. Reading from and writing information to 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**

Where,   
**FILE** is the structure which is defined in the header file <stdio.h>.

## **Opening a file - for creation and edit**

* A file should be opened before any operation is being performed on it.
* The fopen() function is being used for opening the file.

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");

* 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.
* Now let's suppose the second binary file oldprogram.bin exists in the location E:\cprogram. The second function opens the existing file for reading in binary mode **'rb'**.  
  The reading mode only allows you to read the file, you cannot write into the file.

**They can be accessed by using the following modes:**

|  |  |
| --- | --- |
| **Mode** | **Description** |
| “r” | It opens an existing file for reading only. |
| “w” | It opens a new file for writing. If the filename does not exist it will be created and if the file already exists then its contents are deleted. |
| “a” | It appends the existing file. If the filename does not exist it will be created. |
| “r+” | It opens an existing file for reading and writing. It indicates that the file is to be read before writing. |
| “w+” | It opens a new file for reading and writing. If a file with the current filename exists then it is destroyed and a new file name is created. |
| “a+” | It opens an existing file for reading and appending. Its stream is positioned at the end of the file content. |

## **Closing a file**

* The fclose() function is used for closing a file.
* When this function is used the file pointer is disconnected from 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.

* An integer value is returned which will indicate if the function was successful or not.
* In addition to the fclose() function we even have the fcloseall() function which will close all the streams which are open currently except the standard streams (stdin, stdout and stderr).

## **Reading a file**

**Following are the list of functions which are used for reading a file:**

|  |  |  |
| --- | --- | --- |
| **Functions** | **Syntax** | **Description** |
| fscanf( ) | int fscanf (FILE \*stream, const char \*format,....); | It is used for reading the formatted data from the stream. |
| fgets( ) | char \*fgets(char \*str, int size, FILE \*stream); | It stands for file get string. It is used for getting the string from a stream. |
| fgetc( ) | int fgetc (FILE \*stream); | It will return the next character from the stream from the end of the file or an error. |
| fread( ) | int fread(void \*str, size\_t size, size\_t num, FILE \*stream); | It is used for reading data from a file. |

## **Writing a file**

**Following are the list of functions which are used for writing a file:**

|  |  |  |
| --- | --- | --- |
| **Functions** | **Syntax** | **Description** |
| fprintf() | int fprintf (FILE \*stream, const char \* format,...); | It is used for writing the formatted output of the stream. |
| fputs() | int fputs(const char \*str, FILE \*stream); | It is used for writing a line to a file. |
| fputc() | int fputc(int c, FILE \*stream); | It is opposite to fgetc() and is used for writing a character to the stream. |
| fwrite() | int fwrite(const void \*str, size\_t size, size\_t count, file \*stream); | It is used for writing data to a file. |

## **Getting data using 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()

**int fseek(FILE \*stream, long offset, int origin);**

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. |

## **Accepting the command line arguments**

**The main() can accept two arguments**

1. First argument will be an integer value that will specify the number of command-line arguments.
2. The second argument is a full list of all the command-line arguments.  
     
   **Syntax:**  
   **int main (int arg c, char \*argv[])**

Where,  
**arg c** will specify the number of arguments that are to be passed into the program from the command-line including the name of the program.

* argv will contain the list of all the arguments.
* Each element of the array argv is a pointer where each pointer points to a string.
* In main() the command line arguments are accepted by using the argc and argv.

#### **Example**

**Write a program for reading a file character by character and display it on the screen.**  
  
#include <stdio.h>  
#include <string.h>  
void main()  
{  
    FILE \*fp;  
    int c;  
    char fnm[25];  
    printf("\n Enter a filename:");  
    scanf("%s",fnm);  
    fp = fopen(fnm, "r");  
    if (fp==NULL)  
    {  
        printf("\n Error in opening the file");  
        exit(1);  
    }  
    c=fgetc(fp);  
    while(c!=EOF)  
    {  
         putchar(c);  
         c =fgetc(fp);  
    }  
    fclose(fp);  
}

**Output:**  
Assuming that we are using a text file **hello.txt**, that has the following content:  
  
Welcome to TutorialRide  
  
Now, the output is of the above problem will be:  
  
Welcome to TutorialRide

## **Functions used for selecting a record randomly**

**Following are the functions which are used for selecting a record randomly:**

|  |  |  |
| --- | --- | --- |
| **Functions** | **Syntax** | **Description** |
| fseek() | int fseek(FILE \*stream, long offset, int origin); | It is used to reposition a binary stream. |
| ftell() | long ftell (FILE \*stream); | It is used to know the current position of the file pointer. |
| rewind() | void rewind (FILE \*f); | It is used for adjusting the position of the file pointer to the next I/O operation and it will take place at the beginning of the file. |
| fgetpos() | int fgetpos (FILE \*stream, fpos\_t \*pos); | It is used for determining the current position of the stream. |